Canadian Manuscript Report of

Fisheries and Aquatic Sciences 2486

1999

THREE-DIMENSIONAL VISUALIZATION OF FISHERY ACOUSTIC DATA AND BATHYMETRY USING IBM VISUALIZATION DATA EXPLORER

M. Hajirakar[1], R. Kieser[2], R.D. Stanley[2], A.M. Cornthwaite[2], and P.Y. Huang[3]

[1]VGI, Vision Group International

5325 Cordova Bay Road, Suite 211

Victoria, British Columbia, V8Y 2L3

[2]Fisheries and Oceans, Canada

Pacific Biological Station

Nanaimo, British Columbia, V9R 5K6

[3]T.J. Watson Research Center

International Business Machines Corporation

30 Saw Mill River Road

Hawthorne, New York, 10536, USA

Cat. No. Fs97-4/2486E ISSN 0706-6473

Correct citation for this publication:

Hajirakar, M., R. Kieser, R.D. Stanley, A.M. Cornthwaite, and P.Y. Huang. 1999. Three-dimensional visualization of fishery acoustic data and bathymetry using IBM Visualization Data Explorer. Can. Manuscr. Rep. Fish. Aquat. Sci. 2486: 54 p.

ABSTRACT

Hajirakar, M., R. Kieser, R.D. Stanley, A.M. Cornthwaite, and P.Y. Huang. 1999. Three-dimensional visualization of fishery acoustic data and bathymetry using IBM Visualization Data Explorer. Can. Manuscr. Rep. Fish. Aquat. Sci. 2486: 54 p.

Fisheries and Oceans Canada, in collaboration with the Groundfish Research and Conservation Society, conducted an acoustic biomass survey of an aggregation of widow rockfish *(Sebastes entomelas)* off the northwest coast of Vancouver Island, British Columbia, in January 1998. One of the tools used to facilitate the data analysis was three-dimensional visualization of the acoustic data, supported by custom applications developed by Vision Group International (VGI).

VGI used IBM Visualization Data Explorer (DX) software to create the acoustic data visualization. The main steps in producing the visualization involved creating a series of programs to import, select, interpolate and display the acoustic and bathymetry data. Final image manipulation was used to select optimal colours and perspective. Once the acoustic data were examined and imported into DX, ten programs were developed to take the data through a series of steps in order to arrive at the final visualization. The majority of the programs involved organizing and reformatting the acoustic data to prepare them for display in the final image. These programs defined the relationships between different elements of the acoustic data, and applied processes of normalization and/or data screening where required. An important step was the development of an appropriate colour scale to display the fish volume densities. A methodology was also developed for visualizing the acoustic data in three dimensions using a nearest neighbour interpolation technique. The final image was a three dimensional rendering of the acoustic data, consisting of the two-dimensional non-interpolated acoustic data displayed as a curtain, superimposed over the three dimensional interpolated bathymetric surface. The display of the final image was designed to allow the user to control its contents. Through a control panel, the user can specify what components of the image are to be displayed and the amount of data to be displayed. The colour scheme to be applied to the image components is also easily defined.

The three-dimensional visualization procedures developed by VGI rely on data from a Simrad EK500 echosounder. Minor program modifications may be required to process other digital acoustic data sets. Large sets of acoustic data from different time periods can be quickly processed and visualized to gain an understanding of fish density, distribution, movement, and behaviour. The ability to visualize acoustic data has far-reaching implications in many scientific research studies and applications.

RÉSUMÉ

Hajirakar, M., R. Kieser, R.D. Stanley, A.M. Cornthwaite, and P.Y. Huang. 1999. Three-dimensional visualization of fishery acoustic data and bathymetry using IBM Visualization Data Explorer. Can. Manuscr. Rep. Fish. Aquat. Sci. 2486: 54 p.

Pêches et Océans Canada, en collaboration avec la Groundfish Research and Conservation Society, a mené en janvier 1998 un relevé acoustique de la biomasse d'une agrégation de veuve (*Sebastes entomelas*) près de la côte nord-ouest de l'île de Vancouver (Colombie-Britannique). L'un des outils employés pour faciliter l'analyse des données est la visualisation tridimensionnelle des données acoustiques, soutenue par des applications spécialement conçues par Vision Group International (VGI).

VGI s'est servi du logiciel de visualisation IBM Data Explorer (DX) pour produire la visualisation des données acoustiques. La production de la visualisation a été réalisée principalement par la création d'une série de programmes pour importer, sélectionner, interpoler et visualiser les données acoustiques et bathymétriques. On a pour finir traité l'image de façon à obtenir des couleurs et une perspective optimales. Une fois les données examinées et importées dans le DX, dix programmes ont été élaborés pour les soumettre à une série d'étapes en vue d'arriver à la visualisation finale. La plupart des programmes utilisés visaient à organiser et à remettre en forme les données acoustiques pour les préparer en vue de la visualisation dans l'image finale. Ces programmes définissent la relation entre les différents éléments des données acoustiques, et appliquent des procédés de normalization et/ou du criblage des données le cas échéant. L'établissement d'une gamme de couleurs permettant de bien visualiser les densités volumiques de poisson a été une étape importante. Une méthodologie a aussi été mise au point pour visualiser les données en trois dimensions à l'aide d'une technique d'interpolation du plus proche voisin. L'image finale était une représentation tridimensionnelle des données acoustiques, constituée des données acoustiques bidimensionnelles non interpolées superposées comme un rideau à la surface bathymétrique tridimensionnelle interpolée. Le mode de visualisation de l'image finale permet à l'usager d'agir sur son contenu: grâce à un panneau de commande, il peut spécifier quelles composantes de l'image il veut visualiser, et la quantité de données à présenter. La gamme de couleurs à appliquer aux composantes de l'image est aussi facile à définir.

Les procédures de visualisation en trois dimensions élaborées par VGI reposaient sur des données produites par un échosondeur Simrad EK500. Des modifications mineures pourraient devoir être apportées au programme pour permettre le traitement d'autres ensembles de données acoustiques numériques. Il est possible de traiter et de visualiser rapidement de grandes séries de données acoustiques recueillies à des périodes différentes pour déterminer la densité, la répartition, le déplacement et le comportement des poissons. La possibilité de visualiser les données acoustiques va avoir d'importantes retombées dans de nombreuses recherches scientifiques et leurs applications.

TABLE OF CONTENTS

1.0 INTRODUCTION

This report describes the procedures developed for the visualization of acoustic backscatter data that were collected during a rockfish biomass survey conducted by Fisheries and Oceans Canada and the Canadian Groundfish Research and Conservation Society in January 1998. The objective of the survey was to acoustically estimate the biomass of a particular aggregation of widow rockfish (*Sebastes entomelas*), using the CCGS *W.E. RICKER* as the primary acoustic vessel, and assisted by the commercial fishing vessel *FROSTI* (Fig. 1). The survey is described in detail in Stanley et al. (1999).

The purpose of the present report is to summarize the basic procedures we used for employing IBM Visualization Data Explorer software to render three dimensional images of acoustic micro-surveys. The intent is not to provide a comprehensive user's guide. Rather, we aim to simply familiarize users with the application of this visualization software to fisheries acoustic data with the idea that it will aid interested users in making choices as to how to proceed with their particular application. Detailed descriptions of the modules used in this application are provided in Section 5 and in the Appendix. The reader may prefer to omit these sections or only read the overview that precedes the detailed description of each program in Section 5.

2.0 WIDOW ROCKFISH ACOUSTIC DATA

Acoustic data were collected aboard the CCGS *W.E. RICKER* using a SIMRAD EK500 38/120 kHz split beam echosounder (SIMRAD 1993a), and a SIMRAD BI500 data logging and analysis system (SIMRAD 1993b). The *W.E. RICKER* sounded along a predetermined series of parallel, regularly spaced transects, positioned so that they spanned the aggregation, creating cross-sections perpendicular to the aggregation's principal axis (Fig. 1).

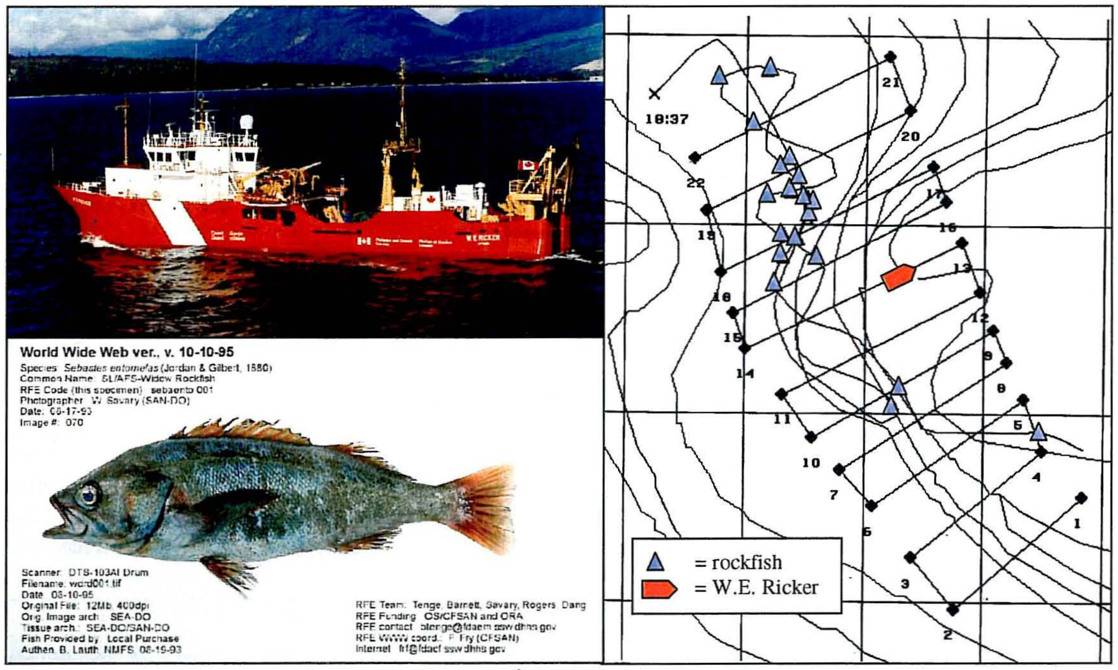

World Wide Web ver., v. 10-10-95
Species: Sebastes entomelas (Jordan & Gilbert, 1880)
Common Name: SL/AFS-Widow Rockfish
RFE Code (this specimen): sebaento 001
Photographer: W. Savary (SAN-DO)
Date: 06-17-93
Image #: 070
Scanner: DTS-103AI Drum
Filename: wcrd001.tif
Date: 08-10-95
Original File: 12Mb 400dpi
Org. Image arch.: SEA-DO
Tissue arch.: SEA-DO/SAN-DO
Fish Provided by: Local Purchase
Authen. B. Lauth, NMFS 08-19-93
RFE Team: Tenge, Barnett, Savary, Rogers, Dang
RFE Funding: OS/CFSAN and ORA
RFE contact: blenge@fdaem.ssw.dhhs.gov
RFE WWW coord.: F. Fry (CFSAN)
Internet: frf@fdacf.ssw.dhhs.gov
18:37
= rockfish
= W.E. Ricker

Fig. 1. The CCGS *W.E. RICKER*, widow rockfish *(Sebastes entomelas)*, and the transect pattern used during the 1998 widow rockfish hydroacoustic survey.

Each complete pass over the set of transects was called a micro-survey. Data were collected at a rate of approximately 1 ping (echo return) per second. For visualization purposes, the data stored on the BI500 were rewritten such that each ping record included log number, date, time, position, bottom depth, and 500 acoustic backscatter values, summarized in 0.5 m intervals over a depth range of 75 to 325 m below the surface.

A sample data set was provided in the form of several ASCII files. For the purpose of developing visualization programs, the sample data set covered only a portion of the total survey area to reduce file size. The sample data were chosen to include typical fish aggregations in the area and covered several transects. The order of the records in the data files was assumed to represent the sequential passage of the vessel along the transects. One file contained acoustic density data, another contained position information, and a third file contained bathymetric data for the area.

The sample acoustic density file contained 2,047 records, corresponding to 2,047 pings (Fig. 2). Each record (ping) contained 500 acoustic backscatter values, resulting in 1,023,500 data values in the entire file. An end of line character marked the end of each record.

The sample position data file also contained 2,047 records, which contained location data for each of the 2,047 pings (Fig. 3). Each record in the position data file corresponded to a record in the same position in the acoustic density file. There were 18 columns in the position data file, of which three were important in the visualization process. These were latitude, longitude, and bottom depth. Latitude and longitude described the geographic location of each ping, and bottom depth was the point at which the bottom was reached by each ping. Three additional columns, which were not processed by the DX programs, describe the depth structure of the data for each ping. Pelagic upper was the depth below the surface at which the acoustic readings started, pelagic lower was the depth below the surface at which the acoustic readings stopped, and pelagic count was the number of acoustic readings taken per ping. For this study, pelagic upper was 75 m, pelagic lower was 325 m, and pelagic count was 500, so that each ping was read at an interval of 0.5 m.

The bathymetric data file contains latitude, longitude, depth, and log values (Fig. 4). The log value is a unique identifier for each record that is assigned by the SIMARAD EK500

```
!bilvgi.1d          .. Pelalig Sv
Sv(500)

    .688E-07     .514E-07     .709E-07     .728E-07     .551E-07     .124E-06     .151E-06
    .759E-08     .161E-07     .288E-07     .645E-07     .295E-07     .531E-07     .905E-08
    .594E-07     .235E-07     .267E-07     .112E-07     .114E-07     .658E-08     .652E-08
    .928E-08     .166E-07     .182E-07     .485E-07     .282E-07     .103E-07     .572E-07
    .329E-07     .504E-07     .466E-07     .455E-08     .689E-08     .184E-07     .321E-07
    .157E-07     .589E-08     .400E-08     .530E-08     .203E-08     .197E-07     .183E-07
    .200E-07     .238E-07     .104E-07     .195E-08     .715E-08     .536E-07     .408E-07
    .269E-07     .333E-07     .110E-07     .254E-07     .343E-07     .545E-07
```

Fig. 2. Subset of the acoustic data file `bilvgi.1d`. Each row has 500 entries.

```
!bilvgi.1p          .. Ping file

Ping     Date&   Time&  Distance!  Latitude!   Longitude!  BottomDepth!  EchogramType&  PelagicUpper!
   1  19980203  90848    365.3000  50.803200  -129.371155        149.00              0          75.00
   2  19980203  90849    365.3000  50.803200  -129.371155        149.80              0          75.00
   3  19980203  90849    365.3050  50.803249  -129.371002        150.70              0          75.00
   4  19980203  90850    365.3050  50.803249  -129.371002        151.00              0          75.00
```

Fig. 3. Subset of the positional data file `bilvgi.1p`, showing first nine columns in the file.

(SIMRAD 1993a). It is not used during the visualization process. There is a total of 2225 records in this data file.

3.0 IBM VISUALIZATION DATA EXPLORER

```
        Long          Lat      Depth          Log
-129.34927     50.77847     174.00     0517.100
-129.34991     50.77778     176.40     0517.150
-129.35127     50.77747     178.00     0517.205
-129.35263     50.77711     188.50     0517.260
-129.35385     50.77665     196.90     0517.315
-129.35385     50.77665     196.90     0517.315
-129.35500     50.77608     205.10     0517.370
```

Fig. 4. Subset of the bathymetry file, `range1.da1`.

IBM Visualization Data Explorer (DX) is a software package for data visualization and analysis. It employs a data-flow driven client-server execution model. The DX server is controlled through a data flow executive, which determines what tasks need to be executed based upon user requests and schedules their execution. The executive can be operated directly via a scripting language, but is more commonly controlled via a graphical user interface (GUI) which can generate code in the scripting language based on the inputs it receives. The DX Prompter is a GUI that allows the user to describe the data files to be used. The Visual Program Editor (VPE) is a GUI in which the user manipulates modules (subroutines or functions) to define the operations that are performed on a data set in order to create the geometric objects which make up an image. The image is usually created with an image module which permits direct 'point-and-click' control over the underlying render, camera, and display modules which produce the image. The user can use a mouse to change the viewing angle of the camera, and alter such parameters as light source, shading, colour, zoom, and rotation.

3.1 The DX data model

DX is based on an integrated, discipline-independent data model that describes and provides uniform access services for any data brought into, generated by, or exported from the software. Thus, DX is easily adaptable to new applications and data. The strength of the DX data model lies in the fact that it not only holds the actual data values of a data set, but also stores the relationships between them. A detailed description of the DX data model can be found in the IBM Visualization Data Explorer's User's Guide (International Business Machines Corporation 1997).

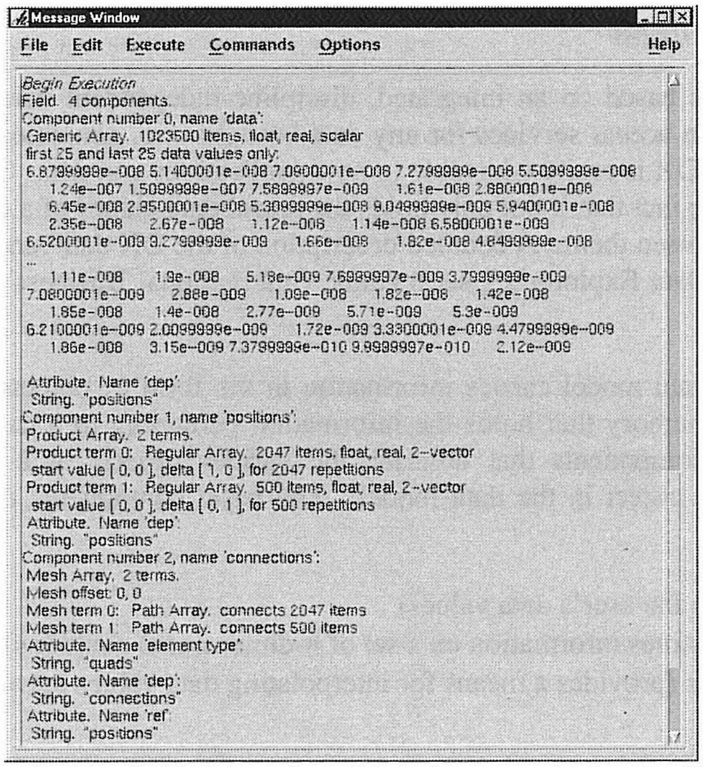
The data model carries information in the form of objects. An object is a data structure stored in memory that holds the information pertaining to a data set. A field object, which consists of components that describe the various aspects of a data set, is the most fundamental type of object in the data model. The basic components of a field object are as follows:

- data (stores the user's data values)
- positions (stores information on a set of n-dimensional positions)
- connections (provides a means for interpolating data values between the positions)

To illustrate, consider the acoustic density data set. This data set contains 2,047 records, and each record contains 500 acoustic density values, in the form of a 2,047 x 500 grid, with 1,023,500 data values in total. Once this data set is imported into DX, it is stored as an array object in the DX data model. The positions and connections are assumed to be regular and are described as product arrays. At this time, the data is associated with a 2-dimensional rectangular grid. The data, positions and connections define a field object.

Fig. 5 outlines the basic components of the field object describing this data set. The data component (component number 0) stores the actual acoustic density values. The first 25 and last 25 data values are listed, along with the total number of items and their data type. The positions component (component number 1) describes the grid format (2,047 x 500) of the data. It stores the two dimensions of the data set (*x*, *y*) in product term 0 and product term 1. Product term 0 declares the *x* component as having 2,047 items, with each item separated by 1 unit (start value [0,0]; delta [1,0]; for 2,047 repetitions). Product term 1 declares the *y* component as having 500 items, with each item separated by 1 unit (start value [0,0], delta [0,1], for 500 repetitions). The connections component connects the 2,047 x 500 grid with 'quad' connections (i.e. connects the items across and down the grid). The connections information can be used to interpolate data values between positions when points are defined on a regular grid.

Another important, but optional, component of a field object is called invalid positions. It is very useful because it allows missing or invalid data to be marked in a data set. The items in this component are either '1's or '0's; a value of '1' indicates a position is invalid, while a value of '0' indicates a position is valid. In the acoustic data file, for example, there is a total of 1,023,500 data values. If the first acoustic data reading for each of the 2,047 records was missing, then these could be flagged in the invalid positions component. This component would also contain 1,023,500 items, with a value of '1' for the 1st, 501st, 1,001st, etc. items and '0's for the remainder. Sections 6.1 and 6.6 discuss this component in more detail.



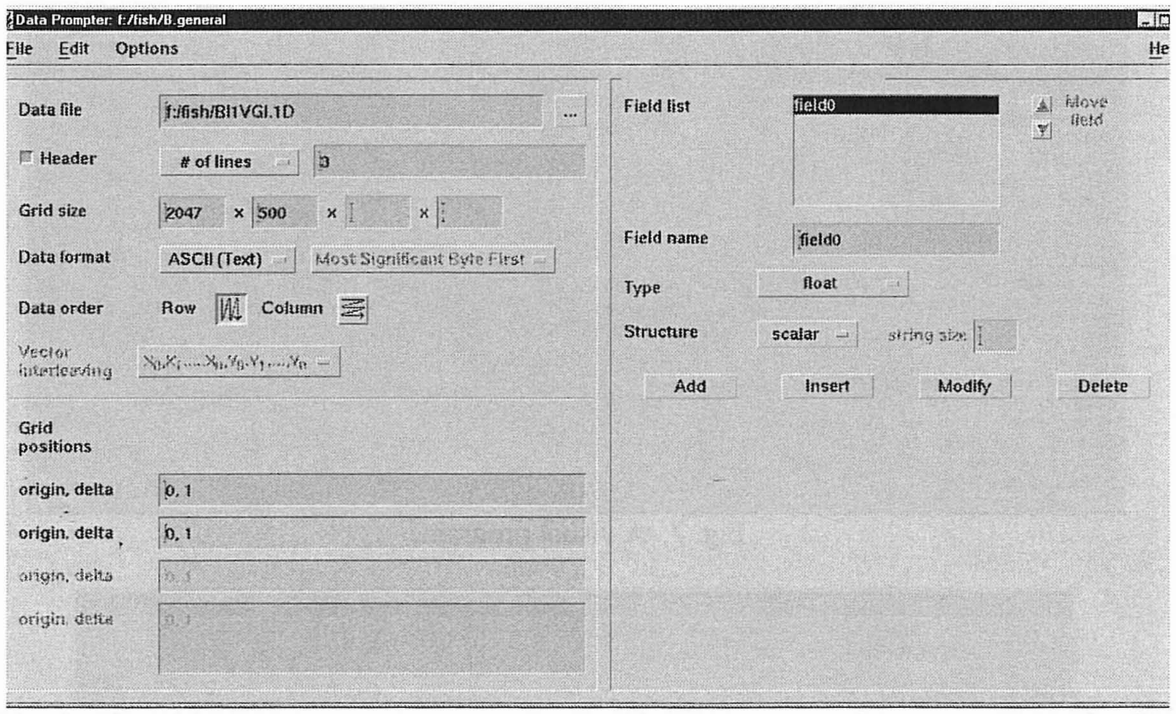
```
Begin Execution
Field. 4 components.
Component number 0, name 'data':
 Generic Array. 1023500 items, float, real, scalar
 first 25 and last 25 data values only:
 6.8799999e-008 5.1400001e-008 7.0900001e-008 7.2799999e-008 5.5099999e-008
     1.24e-007 1.5099999e-007 7.5899997e-009     1.61e-008 2.6800001e-008
     6.45e-008 2.9500001e-008 5.3099999e-009 9.0499999e-009 5.9400001e-008
     2.35e-008     2.67e-008     1.12e-006     1.14e-008 6.5800001e-009
 6.5200001e-009 9.2799999e-009     1.66e-008     1.82e-008 4.8499999e-008
 ...
     1.11e-008     1.9e-008     5.18e-009 7.6999997e-009 3.7999999e-009
 7.0800001e-009     2.88e-009     1.09e-008     1.82e-008     1.42e-008
     1.85e-008     1.4e-008     2.77e-009     5.71e-009     5.3e-009
 6.2100001e-009 2.0099999e-009     1.72e-009 3.3300001e-009 4.4799999e-009
     1.86e-008     3.15e-009 7.3799999e-010 9.9999997e-010     2.12e-009

 Attribute. Name 'dep':
  String. "positions"
Component number 1, name 'positions':
 Product Array. 2 terms.
 Product term 0:   Regular Array. 2047 items, float, real, 2-vector
  start value [ 0, 0 ], delta [ 1, 0 ], for 2047 repetitions
 Product term 1:   Regular Array. 500 items, float, real, 2-vector
  start value [ 0, 0 ], delta [ 0, 1 ], for 500 repetitions
 Attribute. Name 'dep':
  String. "positions"
Component number 2, name 'connections':
 Mesh Array. 2 terms.
 Mesh offset: 0, 0
 Mesh term 0:   Path Array. connects 2047 items
 Mesh term 1:   Path Array. connects 500 items
 Attribute. Name 'element type':
  String. "quads"
 Attribute. Name 'dep':
  String. "connections"
 Attribute. Name 'ref':
  String. "positions"
```

Fig. 5. The field object describing the acoustic density file.

Components can contain attributes. The most common attributes are `dep` and `ref`. The `dep` attribute specifies the component on which a given component depends, while the `ref` attribute specifies the component to which a given attribute refers. In Fig. 5, the data component is dependent on the positions, and therefore has a `dep` attribute of positions. On the other hand, the connections component refers to the positions, and it has a `ref` attribute of positions. The element type is quad and it has a `dep` attribute of connections.

3.2 The DX prompter

The DX prompter is the graphical user interface for the DX General Array Importer, where the structure and contents of a data set are described. The description and a pointer to the actual data file is stored in a `.general` file. Fig. 6 shows the parameters used to describe the acoustic density data file described in Sections 2.0 and 3.1. In this figure, the user has described the acoustic density file `BI1VGI.1D` as a 2,047 x 500 grid. The data type is `float` (floating decimal) and all values are scalar. The data are described in more detail in section 4. Note that the creation of a `.general` file does not perform actual data import; this file will simply be referenced when the data are imported by the user through the VPE.


Data Prompter: f:/fish/B.general

Data file: f:/fish/BI1VGI.1D
Header: # of lines 0
Grid size: 2047 x 500 x x
Data format: ASCII (Text) Most Significant Byte First
Data order: Row Column
Vector interleaving: X0,X1,...,Xn,Y0,Y1,...,Yn
Grid positions
origin, delta: 0, 1
origin, delta: 0, 1
origin, delta: 0, 1
origin, delta: 0, 1

Field list: field0
Field name: field0
Type: float
Structure: scalar
string size: 1

Fig. 6. The DX Prompter interface, populated with a description of the acoustic density file.

3.3 The visual program editor

The operations performed on a data set are defined by a user through the grouping and linking of modules in the visual program editor (VPE). Modules are subroutines (functions) that perform specific operations on objects. As an object flows from one module to another, some modules will add components, some will remove components, and some will change components. A module only operates on the component or components it was designed to handle, and does not affect any other parts of the object. Modules are selected, placed, and connected to each other in the VPE (Fig. 7). All modules contain input and output tabs. Each input tab is an argument, or parameter, for the module. For example, the `Import` module in Fig. 7 exposes three of its many input tabs and one output tab. The first input tab is depressed, indicating that an argument has been specified for the parameter it references. The user has set this argument by double-clicking on the module to bring up the module's configuration dialogue box (Fig. 8). Under the 'Inputs:' heading, there are many rows. Each row refers to an input tab.

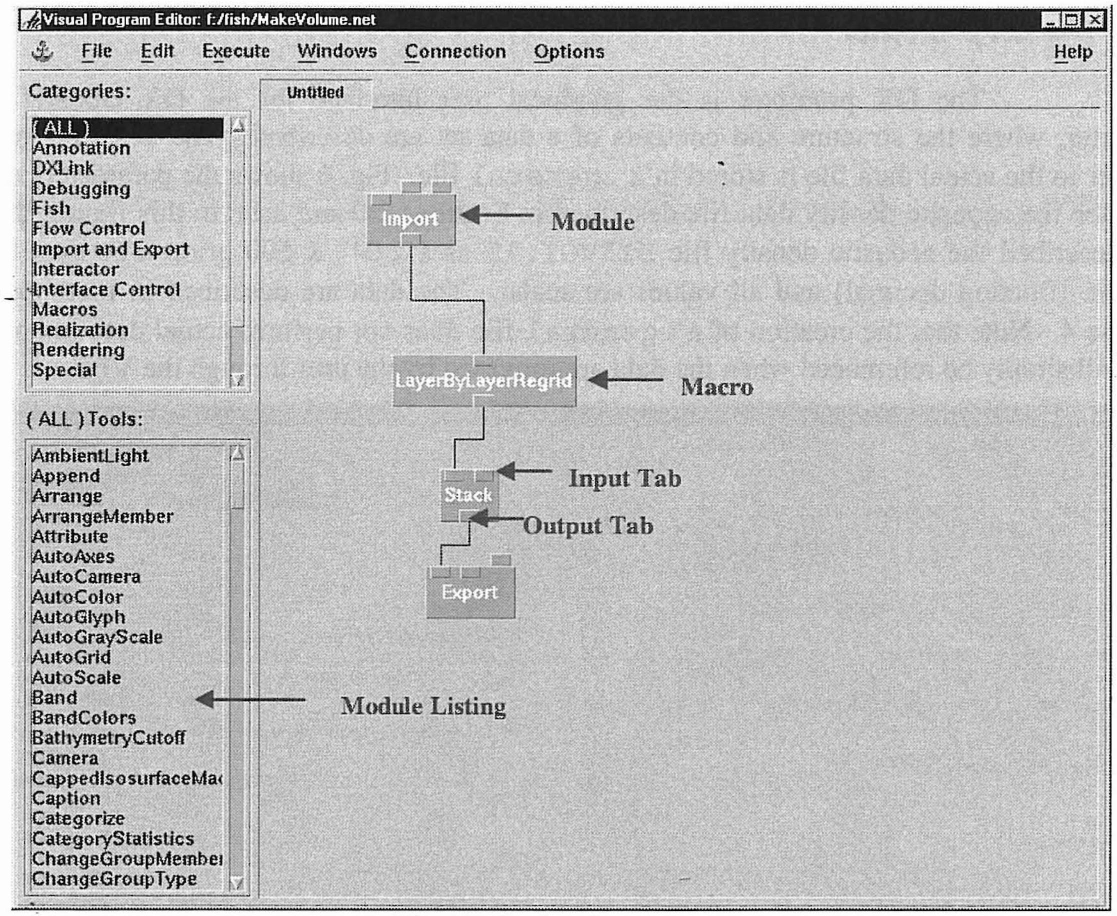
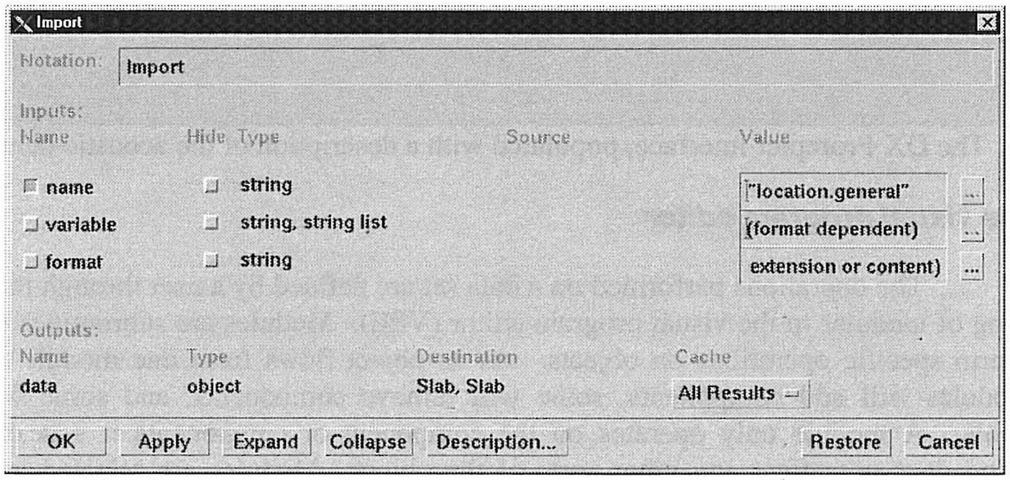


Fig. 7. A visual program.



Notation: Import

Inputs:

Name	Hide	Type	Source	Value
name		string		"location.general"
variable		string, string list		(format dependent)
format		string		extension or content)

Outputs:

Name	Type	Destination	Cache
data	object	Slab, Slab	All Results

Fig. 8. Arguments for each module are entered in a dialogue box

The argument which has been set is referred to by the first input tab, and contains the name of the `.general` file which references and describes the data file being imported.

Interactively setting the input parameters in the configuration dialogue box is one of two ways to specify the arguments for a module. The other method is to simply connect the output of one module to the input tab of another module. For example, the output of the stack module in Fig. 7 is the argument for the first input tab of the export module.

A set of connected modules can be saved as a visual program and has a `.net` file extension. A program can contain one or more macros, which are simply programs that are called from the main visual programs. Macros also have a `.net` extension.

3.4 The Image

DX users can use visual programs to extract, manipulate, and organize the data in a way that is meaningful to the user, and create a 'renderable' geometric object. An image is the result of rendering this object using the DX `Render` module. The `Render` module uses viewing angles provided by the `Camera` module to create a two-dimensional image that represents the three-dimensional renderable object. The two-dimensional image can be displayed on a monitor using the `Display` module, or printed on a printer, or saved to disk. However, because the user cannot interact directly with an image created in this manner, the `Image` module is often used to render images in DX. This module permits the user direct 'point-and-click' control over the underlying `Render`, `Camera`, and `Display` modules which produce the image. The user can customize the final display of an image using a number of mouse-driven interactive tools provided in DX such as `pan/zoom` and `rotate` (Fig. 9).

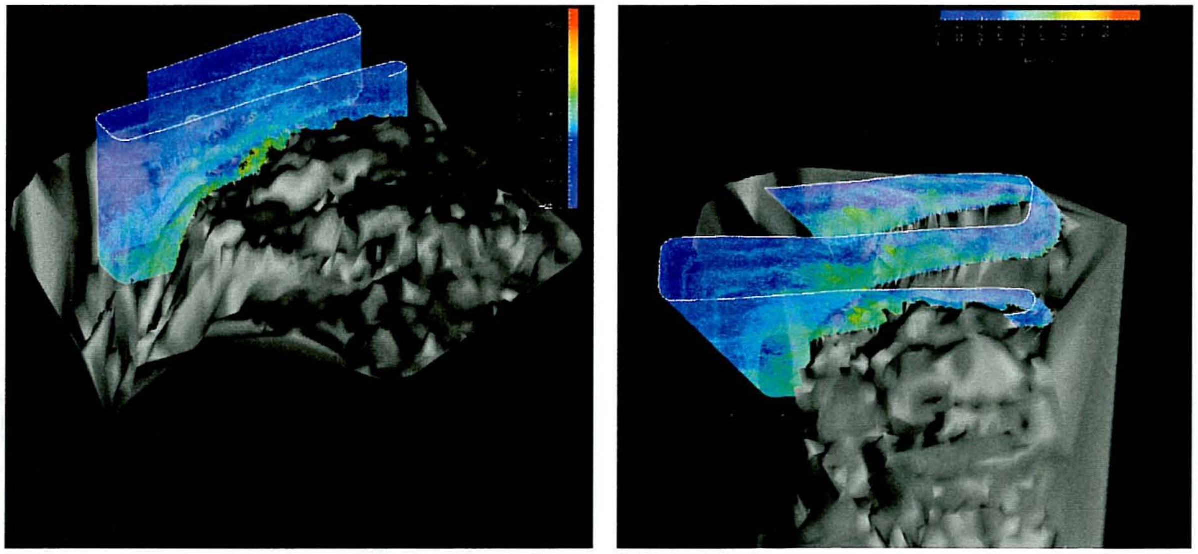


Fig.9. Two views of the same image.

4.0 MODELING ACOUSTIC DATA WITH DX

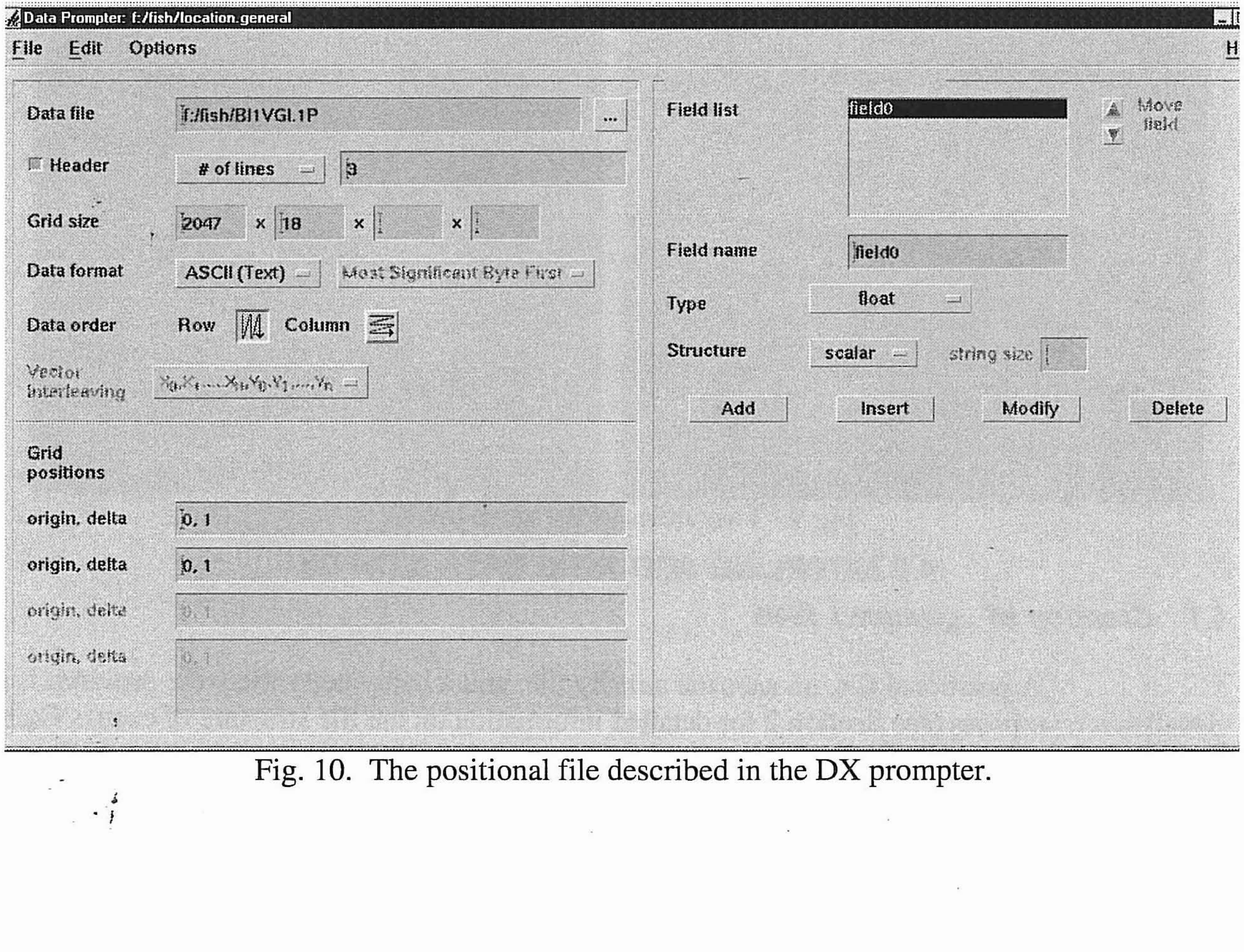
4.1 Creation of `.general` files

A positional file, an acoustic density file, and a bathymetry file were provided for visualization purposes (see Section 2 for detailed information on the file structure of each). Each data file was described in the DX prompter dialogue box. The result of the process were three `.general` files, each containing a pointer to a particular file and a description of its content. These `.general` files can be used by `Import` modules to import the data into DX.

The parameters for the positional file, as described in the DX prompter, are shown in Fig. 10. This data file contains 2,047 records, with 18 columns per record. The full path name of the data file (`f:\fish\BI1VGI.1P`) has been set, along with the dimensions of the grid (2,047 x 18) and the number of lines in the header. The data order has been set to `row`, indicating that each record is contained on a separate row. Default values were used for all other parameters. This description was saved as `Location.general`.

The parameters for the acoustic density file, as described in the DX prompter, are shown in Fig. 11. The description of this file is identical to that of the positional file, with the exception that the *y* dimension of the grid has been set to 500, rather than 18, columns. This description was saved as `B.general`.

The file `Range.general` holds the description and pointer to the bathymetry file (Fig. 12). The name of the data file, including the path, and the number of lines in the header have been set, as for the positional and acoustics data files (Figs. 10 & 11). Unlike the positional and acoustic data, however, the bathymetry data has not been described as a grid. Rather, the first three columns of each record (latitude, longitude, and depth) are described as a three-dimensional vector (3-vector) named locations. The locations field is understood to contain values for positions in three-dimensional space (the *x, y, z* coordinates) for each record. The log field is not used during the visualization process.


Data Prompter: f:/fish/location.general

File Edit Options

Data file: f:/fish/BI1VGI.1P

Header: # of lines 3

Grid size: 2047 x 18 x x

Data format: ASCII (Text) — Most Significant Byte First

Data order: Row Column

Vector interleaving: X0,X1,...Xn,Y0,Y1,...,Yn

Grid positions
origin, delta: 0, 1
origin, delta: 0, 1
origin, delta: 0, 1
origin, delta: 0, 1

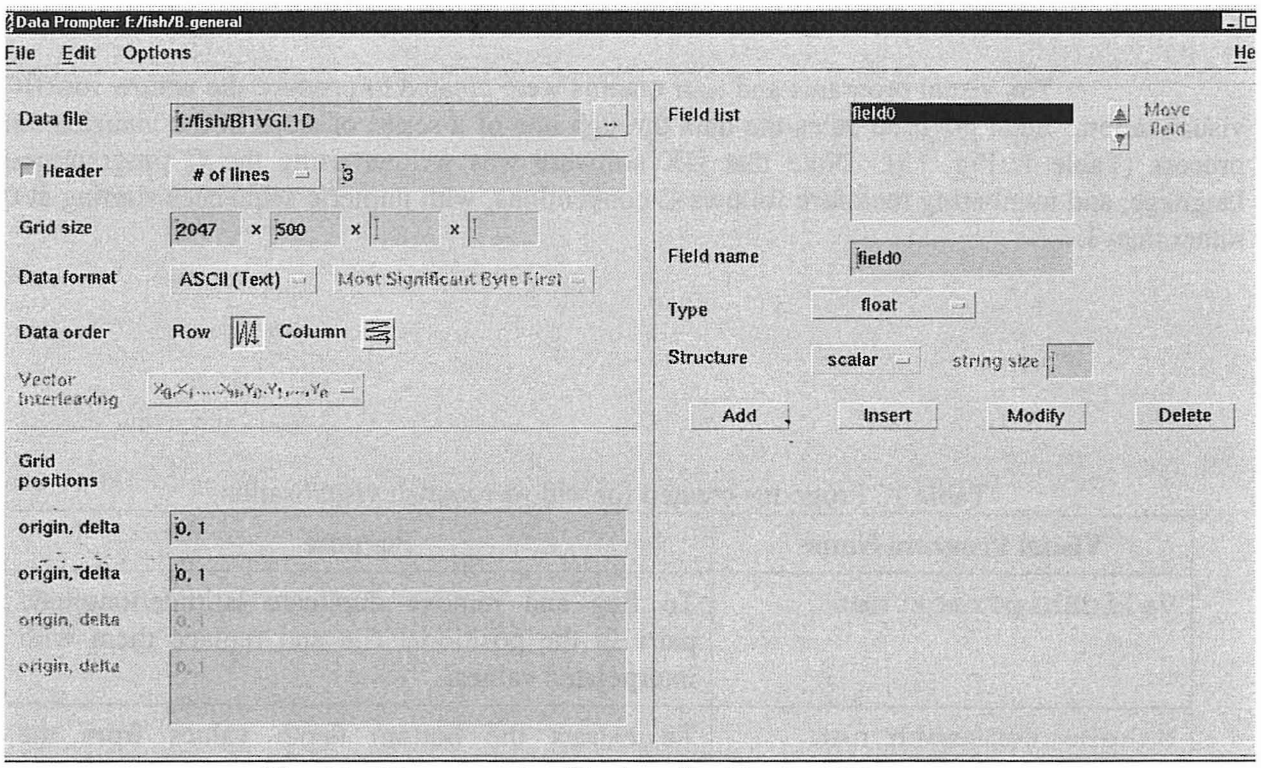
Field list: field0

Field name: field0

Type: float

Structure: scalar — string size

Fig. 10. The positional file described in the DX prompter.


Data Prompter: f:/fish/B.general

File Edit Options He

Data file: f:/fish/BI1VGI.1D

Header: # of lines: 3

Grid size: 2047 x 500 x x

Data format: ASCII (Text) | Most Significant Byte First

Data order: Row | Column

Vector interleaving: $X_0,X_1,...,X_n,Y_0,Y_1,...,Y_n$

Grid positions

origin, delta: 0, 1

origin, delta: 0, 1

origin, delta: 0, 1

origin, delta: 0, 1

Field list: field0

Move field

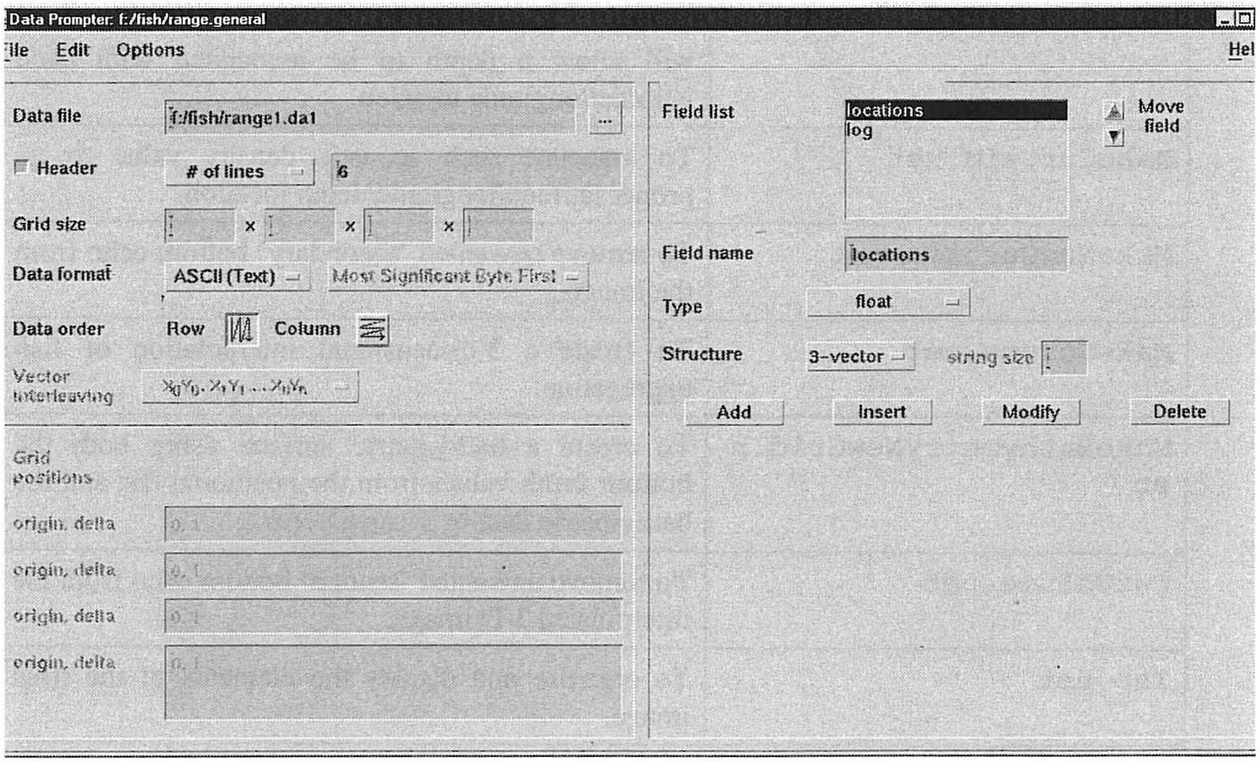
Field name: field0

Type: float

Structure: scalar | string size

Add | Insert | Modify | Delete

Fig. 11. The acoustics file described in the DX prompter.


Data Prompter: f:/fish/range.general

ile Edit Options Hel

Data file: f:/fish/range1.da1

Header: # of lines: 6

Grid size: x x x

Data format: ASCII (Text) | Most Significant Byte First

Data order: Row | Column

Vector interleaving: $X_0Y_0,X_1Y_1,...,X_nY_n$

Grid positions

origin, delta: 0, 1

origin, delta: 0, 1

origin, delta: 0, 1

origin, delta: 0, 1

Field list: locations, log

Move field

Field name: locations

Type: float

Structure: 3-vector | string size

Add | Insert | Modify | Delete

Fig. 12. The bathymetry file described in the DX prompter.

4.2 Overview of Visual Programs

Ten visual programs and four macros were created to produce the widow rockfish visualization. Each program takes the data through one of a series of steps in the visualization process (Table 1, Fig. 13). Note that DX software was written using the C programming language, and numbering therefore follows C conventions, with numeric sequences starting at 0, rather than 1.

Table 1: Programs created for widow rockfish visualisation.

Visual Program Name	**Purpose**
`ValidShipTrack.net`	To flag and remove duplicate latitude/longitude pairs in the positional file and replace them with interpolated values.
`MakeBottomDepth.net`	To extract the bottom depth values from the positional file.
`MakeNewBottomDepth.net`	To replace bottom depth values of '0' with interpolated values.
`ShipTrack3D.net`	To create a 3-dimensional positional structure that will allow a depth to be associated with each latitude/longitude location.
`MakeCurtain.net`	To associate each acoustic density value to its proper latitude/longitude/depth location.
`MakeNewCurtain.net`	To remove unwanted 'secondary' bottom echo from the image.
`MakeVolume.net`	To create a 3-dimensional interpolation of fish aggregation.
`MakeBathymetryNewGrid.net`	To create a bathymetric surface using both the bottom depth values from the positional file and the bathymetric data in `range1.da1`.
`CutVolume.net`	To remove unwanted 'second' bottom echo from the interpolated 3-D image.
`Iso.net`	To organise and display the elements of the final image.

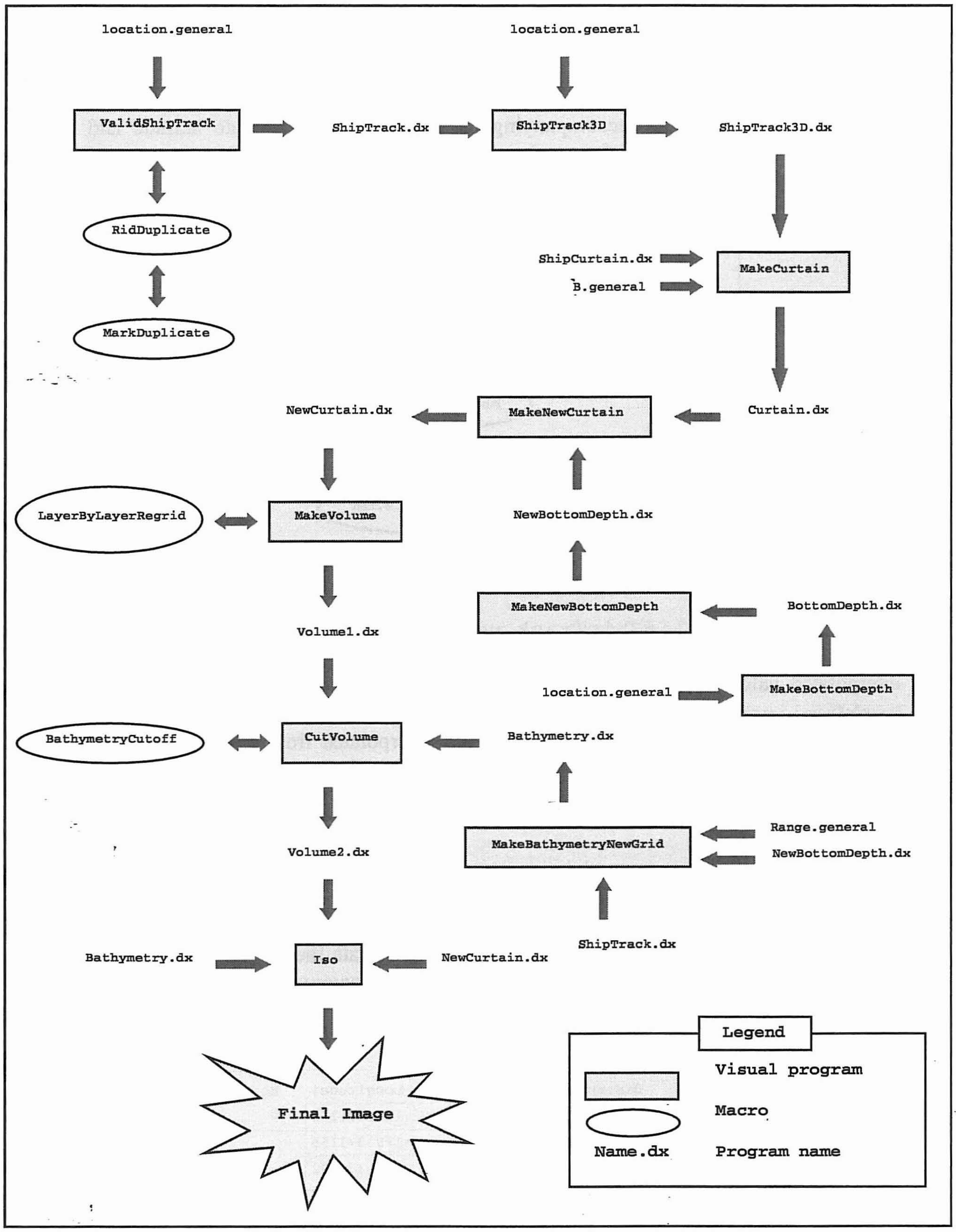


Fig. 13. Overview of the visualisation process.

5.0 VISUAL PROGRAM DESCRIPTIONS

5.1 The `ValidShipTrack.net` program

Duplicate latitude and longitude values occur occasionally in the positional data file when the GPS receiver is not updated between pings. This results in a vector of 500 backscatter values with no accompanying unique location. Duplicate latitude and longitude values can be removed from the positional data file using a program called `ValidShipTrack.net`. This program flags and removes duplicate latitude and longitude values and replaces them with interpolated values (Figs. 14 & 15).

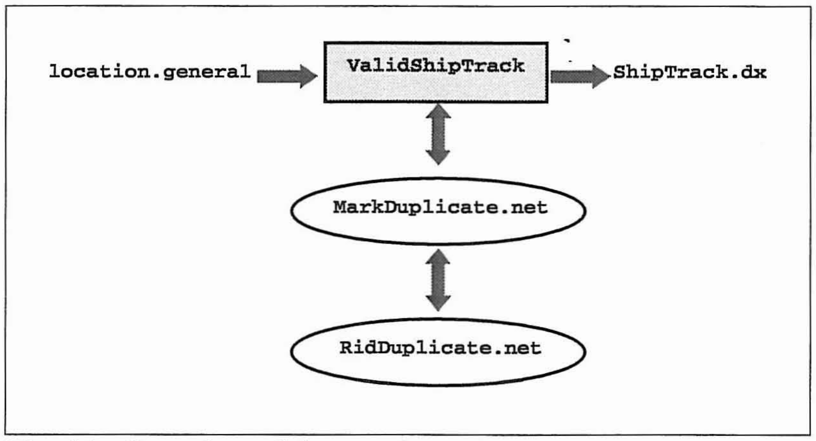


Fig. 14. Overview of the `ValidShipTrack.net` program.

The `ValidShipTrack.net` program has two sections (Appendix Fig. 1). Section A normalizes the latitude and longitude data and replaces the duplicate values with interpolated values using two macros called `RidDuplicate.net` (Appendix Fig. 17) and `MarkDuplicate.net` (Appendix Fig. 18). Duplicate latitude and longitude values are flagged as invalid and replaced by new values interpolated from the adjacent unique values. Section B combines the latitude/longitude pairs into a single field which is then exported.

Section A.

An `Import` module imports `location.general` and creates a field object. The latitude and longitude columns are extracted from the data file using two `Slab` modules with the third input tab of each set to position=4 and position=5, respectively, corresponding to the fifth (latitude) and sixth (longitude) columns of the data file. Each `Slab` module outputs a field object (field) containing either latitude or longitude that is then sent to the `RidDuplicate.net` macro. This macro runs twice, producing a normalized data set, with

```
!BI1VGI.1P          .. Ping file

Ping      Date&    Time&    Distance!    Latitude!    Longitude!    BottomDepth!    EchogramType&
   1   19980203    90848    365.3000    50.803200   -129.371155          149.00                0
   2   19980203    90849    365.3000    50.803200   -129.371155          149.80                0
   3   19980203    90849    365.3050    50.803249   -129.371002          150.70                0
   4   19980203    90850    365.3050    50.803249   -129.371002          151.00                0
```

Fig. 15. Example of repeat locations in the positional file.

the duplicate values removed and replaced by interpolated values, for each field (Appendix Fig. 2).

Section B.

Two `Extract` modules extract the data components of the normalized latitude and longitude fields and send them to a `Compute` module which creates an array of two-dimensional (*x*, *y*) vectors. This array is sent to a `Construct` module which creates a positions component and one-dimensional line connections for the field. This field is output with an `Export` module and called `ShipTrack.dx`.

5.1.1 The `RidDuplicate.net` macro

The `RidDuplicate.net` macro has three parts and runs once each for the latitude and longitude fields at the end of Section A of the `ValidShipTrack.net` program (Appendix Figs. 1 & 2). The process of removing duplicate values is called normalization. This macro normalizes each field by removing the duplicate values and then replaces them with linearly interpolated values.

Part I.

An `Input` module sends the field to an `Extract` module, where the data component is extracted and sent to both an `Inquire` module and to the second input tab of the `MarkDuplicate.net` macro. The `Inquire` module determines the number of items in the data component and, for the sample data set, returns the value '2,047'. A `Compute` module executes the expression *a-1*, subtracting 1 from the result of `Inquire`, and sends the result to the first input tab of the `MarkDuplicate.net` macro. The `MarkDuplicate.net` macro checks for duplicates and returns a list of '1's and '0's indicating which records are duplicates ('1's) and which are unique ('0's). The `MarkDuplicate.net` macro compares each latitude or longitude value to the value immediately following; therefore, the first value will always be marked as unique, while the last value of the data set can be a duplicate.

Part II.

The output from `MarkDuplicate.net` is sent to the `Options` module, which adds a `dep` attribute to force the data component of these values to be dependent on the positions component. The output from `Options` is an array with a `dep` attribute on the positions component, but it is not yet associated with any positions. A `Compute` module changes the number format from integer to byte. The resulting field is sent to the `Replace` module, whose function is to take a component from a source field and place it in a destination field. In this case, the source field is the data component of the list of '1's and '0's that has been processed by the `Options` and `Compute` modules. The destination field is sent from the `Input` module, and has a data component containing the latitude or longitude values, as well as positions and connections components. The data component from the source field is placed in a new component in the destination field called invalid positions, thereby marking the duplicate latitude or longitude values as invalid. The output from `Replace` is sent to an `Include` module with the 'cull' input tab set to '-1', where the invalid records are removed from the data set. The field is now normalized and, for the sample data set, contains 1,014 unique longitude values or 1,002 unique latitude values in the data component, and a corresponding number of unique values representing the unique positions in the positions component. The values in the positions component are extracted with an `Extract` module and sent to a `Compute` module where the *x* component is extracted from the position vector, making it a scalar.

Part III.

The output from the `Compute` at the end of Part II, values representing the unique positions in the positions component of the sample data set, is sent both to an `Inquire` module and to the first input tab of a `Select` module. The `Inquire` module determines the number of items in the data component of this data set. The output of `Inquire` is '1,002' for latitude and '1,014' for longitude, and is a scalar value representing the number of unique values in each sample data set, since the invalid (duplicate) items have been removed. Note that due to the numbering conventions used in DX, these items are counted starting at '0', so that position '1,001' in the sample latitude data set, and position '1,013' in the sample longitude data set, correspond to the last items in each.

The output of `Inquire` is sent to a `Compute` module which executes the expression *a-2*. This is because we want to drop the last unique position in the list and replace it with the last position value from the input data with `location.general` header file so that our interpolated points cover the whole ship track. This results in a value of '1,000' for latitude and '1,012' for longitude, which is sent as the second input tab ('end') of an `Enumerate` module. The first input tab ('start') of the `Enumerate` module contains the value '0', the second input tab ('end') contains the output from `Compute`, and the third input tab ('delta') contains the value '1'. The `Enumerate` module creates a list of numbers beginning at 'start' and ending at the value sent from `Compute`, with increments of 'delta' between each item. This list corresponds to the positions of the items in the data set of unique latitude or longitude values with the last item removed.

The duplicate values were determined by the `MarkDuplicate.net` macro by comparing each value to the value immediately following. Therefore, the first longitude or latitude value in the data set will always be maintained, but the last value will be removed if it is a duplicate of the second to last value. However, in order for the interpolation to proceed properly, it is imperative that the very first and last positions of the data set are maintained. This ensures that the duplicate values always have one value preceding and one value following from which new values can be interpolated. In order to keep the last value and still maintain the actual number of unique values, the last position will be removed from the normalized data set in order to 'make room' for the very last position from the original data set.

The `Select` and `List` modules perform the task of discarding the last unique position and replacing it with the position immediately before the last position. For the sample data set this is the 2,046th position. The output of `Enumerate` is sent to the second input tab ('items to select') of the `Select` module. The first input tab of `Select` ('object to select from') is the one-dimensional vector of unique positions sent by the `Compute` module at the end of Part II. The Select module uses the list of position numbers from Enumerate to select the corresponding values from the vector of unique positions. The last item, corresponding to position 1,001 or 1,013, respectively, of each data set is not selected, because the last item in the list from Enumerate is '1,000' or '1,012', respectively. There are now 1,001 unique positions corresponding to the sample latitude data set, and 1,012 unique positions corresponding to the sample longitude data set. The Select module sends this set of positions to the first input tab of to a List module. The second input tab contains the value '2,046' received from the Compute (*a-1*) module in Part I, and corresponds to the position of the last value in the original latitude or longitude data set. List appends the value '2,046' to the list of unique positions, resulting in a list of 1,014 or 1,002 positions.

An Extract module receives the latitude or longitude data set with the duplicate values removed from the Include module in Part II, and sends the data component to the fourth input tab ('data') of a Construct module. The first input tab ('positions') of this Construct module contains the output of List which is a list of valid position numbers. The output of Construct is a field of irregular positions with regular connections, and the data component contains the unique positions and their associated latitude or longitude values, with the last unique value removed and replaced by the last value from the original data set. This field is sent to the second input tab ('map to be used') of a Map module. An additional Construct module receives the value '2,046' from the Inquire module in Part I. The first input tab ('origin') of this module is set to '0', while the second ('delta') is set to '1'. The output is a field with regular connections which has a positions component consisting of 2,047 positions, starting at '0', and ending at '2,046'. This field is sent to the first input tab ('field to be mapped') of the Map module.

The Map module steps through each position in the 'field to be mapped' and looks up the corresponding position in the 'map to be used'. This means that each position number in the field containing 2,047 positions is looked up in the positions component of the field containing the unique positions and data values. If a corresponding position exists, the data value (longitude or latitude) for that position is associated with it. If a corresponding position does not exist, a data value is interpolated for that position. The procedure is a linear interpolation that utilizes the unique value immediately before and immediately after any series of duplicate values to calculate appropriate new values that are equally spaced between the two original unique values. For example, if there were only one duplicate value, the replacement value would be the average of the two neighbouring unique values. The 2,047 positions are all connected, so that when a corresponding position is not found in the 'map to be used' then the nearest connected position that does contain corresponding positions and data in the 'map to be used' will be used to interpolate a new data value. The result is a field that contains unique longitude or latitude values for each of the 2,047 positions. This field is returned to the ValidShipTrack.net program.

5.1.2 The **MarkDuplicate.net** macro

At the end of part I of the RidDuplicate.net macro, output is sent to MarkDuplicate.net (Appendix Fig. 3), which checks the data for duplicates and returns a vector of '1's and '0's indicating which records are duplicates ('1's) and which are unique ('0's).

The first Input module contains the data component (latitude or longitude values) output from the Extract module in part I of the RidDuplicate.net macro. This is sent to two Select modules as the first input tab of each. The second input tab contains a value specifying which item of the data component to select. These instructions are obtained via the second Input module and the ForEachN/Compute modules.

The second Input module contains the value '2,046' output from the Compute (*a-1*) module in part I of the RidDuplicate.net macro, which is the position number of the last (2,047th) value in the original data set. This value is sent to the ForEachN module, which initiates an iteration starting at '1' and ending at '2,046'. Each count is sent both to a Compute module and to the second Select module. The Compute module subtracts 1 from each count, effectively initiating a new iteration which starts at '0' and ends at '2,045'. Each count from this iteration is sent to the first Select module. The iterations sent to each Select module are the instructions which specify which record to select for comparison. Therefore, as the count sent to the first Select module ('0' to '2,045') is always one count below that sent to the second Select module ('1' to '2,046'), each latitude or longitude value selected from the data set will always be compared to the value immediately following.

The output of both Select modules is sent to a Compute module, which runs a check to see if the first value is equal to the second value. If this is true, then a value of '1' is returned, while if it is false, a value of '0' is returned. This comparison takes place for each pair of counts received by the Select modules. The List, GetLocal and SetLocal modules then work together to create a list of these '1's and '0's (Table 2). Each time one iteration is completed, GetLocal and SetLocal work in tandem to place and retrieve the list from the cache, while List adds actual values to the list. The 'initialize' input argument in GetLocal is set to '0', thereby ensuring that only the last value of any set of duplicates is flagged with a '1'. Once all items in the data component have been processed, this list is sent back the RidDuplicate.net macro.

5.2 The MakeBottomDepth.net program

The MakeBottomDepth.net program extracts the bottom depth values from the positional file using three modules called Import, Slab, and Export (Fig. 16 & Appendix Fig. 4).

The Import module imports location.general and sends it to a Slab module, which extracts the bottom depth column (position=6), making a new field with a data component consisting of only bottom depth. The connections are now connecting 2,047 x 1 positions. This field is output with the Export module and called BottomDepth.dx.

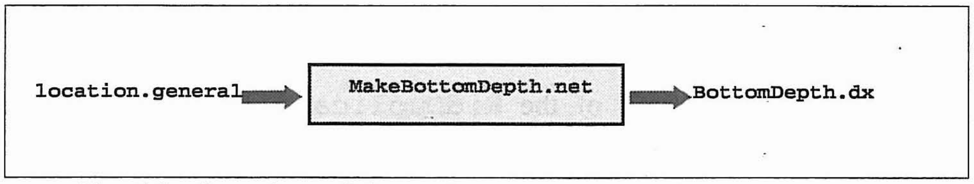


Fig. 16. Overview of the MakeBottomDepth.net program.

5.3 The MakeNewBottomDepth.net program

Occasionally the acoustic logging equipment fails to detect the bottom depth and instead records a zero in the data file. The MakeNewBottomDepth.net program replaces these values with interpolated values by a process similar to the normalization and interpolation process in the ValidShipTrack.net program (Fig. 17).

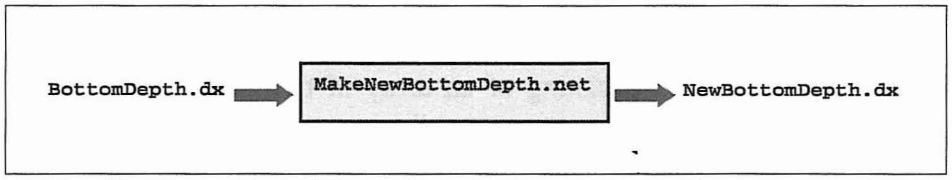


Fig. 17. Overview of the MakeNewBottomDepth.net program

The MakeNewBottomDepth.net program consists of three sections (Appendix Fig. 5). Section A creates a one-dimensional array containing all depth values, Section B removes all the zeros, and Section C interpolates the missing values and exports the new field.

Section A.

An Import module sends BottomDepth.dx to two modules, Extract and Inquire. The Extract module sends the data component to the fourth input tab ('data') of a Construct module. The Inquire module determines the number of positions in each dimension and returns a two- dimensional vector (a 2-vector) with a value of [2,047 1]. This 2-vector is then sent to a Compute module, which extracts the first component ('2,047'), and sends this number to the third input tab ('count') of the Construct module. The Construct module creates a field using the inputs in 'data' and 'count' as arguments. The resultant field contains a positions component that describes a one-dimensional array of 2,047 items, a connections component that connects the items, and a data component consisting of bottom depth values.

Section B.

The output from Construct is sent to an Include module, which selects all bottom depth values greater than 10.0, thereby excluding the zeros. Two Extract modules extract the positions and data components from the selected set and sends them to an additional Construct module, which creates a new field containing only the non-zero bottom depth data and their associated positions.

Section C.

A Map module receives the output from Sections A and B. The field containing all depth values output from Section A is fed into the first input tab ('input') of Map, while the second input tab ('map') receives the field containing only the non-zero depth values from Section B. Each item in the positions component of the input field (all depth values) is looked up in the positions component of the map field (non-zero depth values). The corresponding bottom depth values are looked up, and values are linearly interpolated from the neighbouring values for those values that were originally zero, in the same way that positions were interpolated in the RidDuplicate.net macro. The interpolated values are used to replace the zeros in the data component of the input field. The result is a field with 2,047 positions and a non-zero bottom depth value for each position, called NewBottomDepth.dx.

5.4 The ShipTrack3D.net program

The field ShipTrack.dx, output from the ValidShipTrack.net program, has a data component consisting of an array of two dimensional vectors (2-vectors) containing latitude and longitude values corresponding to each ping. The ShipTrack3D.net program creates a three-dimensional positional structure that associates depth with each latitude/longitude location by declaring the 2-vector field as a 3-vector field (Fig. 18 & Appendix Fig. 6).

ShipTrack.dx is imported by an Import module and sent to an Extract module, where the positions component is extracted from the field. A Compute module declares the latitude/longitude pairs which are the data component of this field as 3-vectors using the expression *a.x, a.y, 0*. An Export module exports this field as ShipTrack3D.dx., an array with three columns, *x*, *y*, *z*, where *x* = the longitude values, *y* = the latitude values, and *z* = the depth values (all zeros). This field shows the ship track on the surface of water (depth=0).

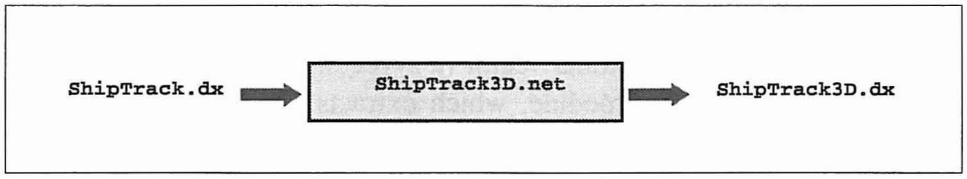


Fig. 18. Overview of the ShipTrack3D.net program

5.5 The MakeCurtain.net program

The MakeCurtain.net program associates each acoustic density value to its proper latitude/longitude/depth location. By displaying the connections between these data points and shading the resultant image, the program creates a 'curtain' of depth values corresponding to the path of the acoustic beam as the vessel moves along the transects (Fig. 19). The header file ShipCurtain.dx describes the three-dimensional positional structure of the widow rockfish acoustic data (Fig. 20). This file is created in a text editor, and defines five object definitions, with each definition consisting of various clauses. A total of five objects are defined, with the first four objects used to define the last object (Table 3).

The MakeCurtain.net program imports the files ShipCurtain.dx and B.general using two Import modules (Appendix Fig. 7). The positions and connections information for the acoustic density data is contained in ShipCurtain.dx, while

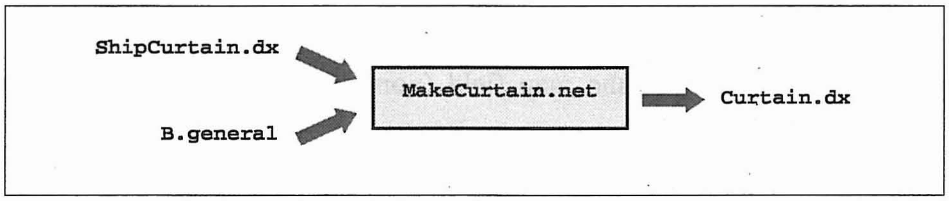


Fig. 19. Overview of the MakeCurtain.net program.

```
object  1  class  array  type  float  rank  1  shape  3  items  2047  data  file
ShipTrack3D
object 2 class  regulararray count 500
origin 0.0 0.0 -75.0
delta 0.0 0.0 -0.5
object 3 productarray
     term 1
     term 2
object 4 class gridconnections counts 2047 500

object 'field' class field
component 'positions' 3
component ' connections' 4
end
```

Figure 20. The `ShipCurtain.dx` header file.

Table 3: Object descriptions in ShipCurtain.dx

Object Name	Description
Object 1	Defines an array of 2047 items of type float, 3-D vector; with longitude, latitude and depth(=0). The data used to populate this array are from the array file ShipTrack3D (the result of ShipTrack3D.net)
Object 2	Defines a regular array with 500 items, with an origin of (0.0, 0.0, -75.0) and a delta of (0.0, 0.0, –0.5); this array captures the structure of the depth component of the locational data.
Object 3	Defines a product array composed of 2 terms: Term 1: refers to Object 1 Term 2: refers to Object 2 Therefore, this object contains both objects 1 and 2; each latitude/longitude pair with is associated with 500 depth values
Object 4	Defines a set of 2047 x 500 connections on a regular grid
Field	Defines a Field Object, consisting of positions from Object 3 and connections from Object 4

`B.general` describes and points to the actual acoustic density file. This information is merged using a `Replace` module, which replaces the positions component from `B.general` with the positions component from `ShipCurtain.dx`. The resultant field contains both the acoustic densities (data component) and the locations (positions component), so that each acoustic density value is associated with a latitude/longitude/depth location. An `Export` module exports this field as `Curtain.dx`.

5.6 The `MakeNewCurtain.net` program

Acoustic data were collected for depths ranging from 75.0 m to 324.5 m below the surface. Backscatter information for the entire range was recorded in the data file. For locations where the bottom depth, defined as the substrate-to-water interface, was shallower than 324.5 m, the acoustic density file contained data from below the bottom. This is due both to penetration of the substrate by the acoustic beam, and to multiple reflections from the surface and the bottom that can lead to a second and higher bottom echo. Including the backscatter data from below the bottom tends to obscure the final image, especially when it is rotated to different angles. Therefore, for the purpose of creating the final image, we considered any acoustic backscatter data originating below the bottom to be 'nonsense' values which should be removed from the data set. These 'nonsense' values can be removed using the MakeNewCurtain.net program, which 'cuts off' the 500 cell vector of acoustic density data where it reaches the actual bottom depth (Fig 21).

The MakeNewCurtain.net program has three sections (Appendix Fig. 8). Section A marks the acoustic density data from Curtain.dx as valid or invalid by comparing each depth to the depths in the bottom depth file MakeNewBottom.net. Section B adds an invalid positions component to the Curtain.dx field and exports it as NewCurtain.dx. Section C collects information from Sections A and B to be presented in the image.

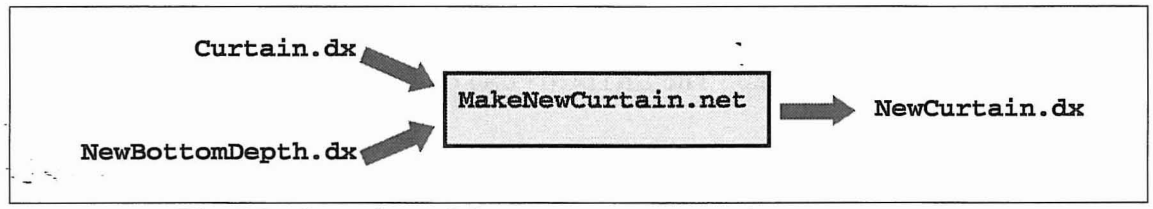


Fig. 21. Overview of the MakeNewCurtain.net program.

Section A.

The files NewBottomDepth.dx and Curtain.dx are imported using two Import modules. These files are the output of the MakeNewBottomDepth.net and MakeCurtain.net programs. The bottom depth values from NewBottomDepth.dx are made negative using a Compute module and then sent to the second input tab ('Ocean Depth') of the ValidDepth module. An Extract module extracts the positions component from Curtain.dx, and sends the depth values to the first input tab ('Position Array') of the ValidDepth module. The ValidDepth module is a specially written runtime loadable module for this project. It is easily created using the Module Builder tool that comes with the standard DX tool kit. This module 'cuts off' the 500 cell vectors with the bottom depth values and compares each bottom depth value from the array in the 'Ocean Depth' input to each of the 500 depth values in the corresponding item from the 2,047 x 500 'Position Array' input. If the acoustic sample is above bottom, a value of '1' is put into an array. If the acoustic sample is below bottom, a value of '0' is put into the array. The output of the ValidDepth module is an array of 2,047 x 500 items, where the items consist of depth indicators in the form of '0's and '1's. Note that although '0's and '1's represent 'valid' and 'invalid' items, respectively, these values are actually reversed because the bottom depths are negative.

Section B.

The output from the ValidDepth module is sent to an Options module, which adds a dep attribute to force the data component of these values to be dependent on their positions component. This ensures that each position is associated with either a valid or invalid indicator. A Compute module is used to reverse the '0's and '1's to account for the negative depth values input into the ValidDepth module. A second Compute module changes the number format from integer to byte because the invalid positions component uses byte format. The byte data then is passed to the Options module to add a dep attribute to the positions component. The resulting field is sent to the Replace module, whose function is to take a component from a source field and place it in a destination field. In this case, the source field is the data component of the list of '1's and '0's that has been processed by the Options and Compute modules. The destination field is sent from the Input module, and has a data component containing acoustic density data and a positions component containing latitude/longitude/depth data. The data component from the source field is placed in a new component in the destination field called invalid positions, thereby marking the acoustic backscatter values associated with depths below the bottom as invalid. The export module exports this field as NewCurtain.dx.

Section C.

The field NewCurtain.dx from Section B is sent to an Include module, which selects the data which has been marked as valid, and passes it to a Compute module. Compute applies a log scale to the data and sends it to the AutoColor module, which applies a standard blue to red colour scheme to the data. The output of AutoColor is sent to the first input tab of the Collect module, and represents the 'curtain' of data, corresponding to the path of the acoustic beam above the ocean floor as the vessel moves along the transects.

The Slab module receives the Curtain.dx field from the Import module in Section A and extracts the first column, corresponding to the 'top' of the curtain and representing the surface ship track. ShowConnections draws the connection elements for the extracted data, and sends the output to the Color module which applies the colour red to the drawing. The output of Color is sent to the second input tab of the Collect module and represents the 'ship track.' The Collect module combines the 'curtain' image with the 'ship track' image and sends the output to the Shade module which applies a smooth shading to the image based on the quad connections between the data points, assuming a lighting source at the camera position. The Scale module reduces the height of the image before it is displayed so that the final image fits in a rectangular display window with a reasonable aspect ratio (Fig. 22).

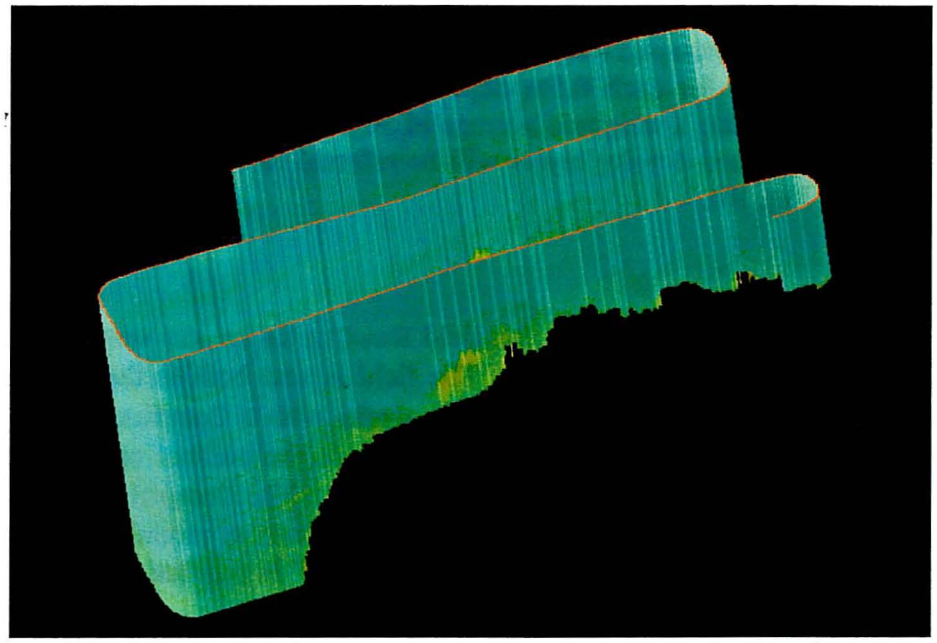


Fig. 22. Image of 'curtain' cut off by the bottom depth values.

5.7 The `MakeVolume.net` program

The `MakeVolume.net` program employs a distance weighted interpolation to create a three-dimensional image of the fish aggregation (Fig. 23 & Appendix Fig. 9). The interpolation method involves 'stacking' 500 horizontal 40 x 40 grids in the *z* dimension, from the top to the bottom of the sample range (-75 m to -324.5 m), with 0.5 meters between each grid. The interpolation uses a simple nearest neighbour logic. Each 40 x 40 grid has 41 x 41 vertices, and each vertex is assigned a value based on a weighted average of the 30 nearest points within a radius equal to three times the spacing between the vertices. The weighting is inversely related to the distance. The interpolation for each grid is in two dimensions, but by 'stacking' the 500 interpolated grids in the *z* dimension, a three-dimensional volume effect is created.

The `MakeVolume.net` program imports `NewCurtain.dx` with an `Import` module, which sends the field to the `LayerByLayerRegrid.net` macro (Appendix Figs. 9 & 10). This macro creates the grids and performs the interpolations. The resultant grids are 'stacked' in the *z* dimension using the `stack` module, and exported with an `Export` module as `Volume1.dx`.

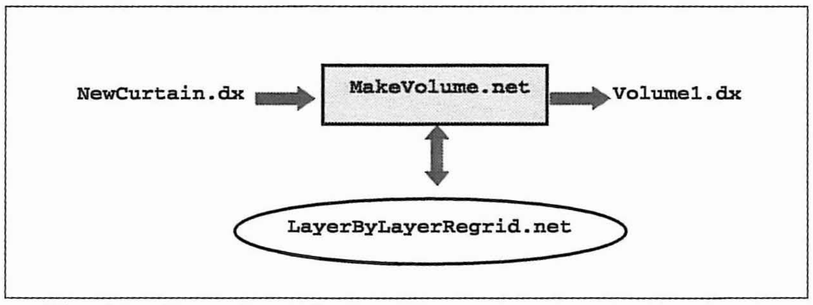


Fig. 23. Overview of the `MakeVolume.net` program.

5.7.1 The `LayerByLayerRegrid.net` macro

The `LayerByLayerRegrid.net` macro creates a 40 x 40 rectangular grid that covers the survey area for each 0.5m depth increment in the acoustic density file, and performs an interpolation using a weighted average of 30 nearest neighbours to populate the cells in each grid. The interpolated density layers will be referred to as isosurfaces. The macro has three parts (Appendix Fig. 10).

Part I

An `Input` module accepts the field `NewCurtain.dx` sent from the `Import` module of the `MakeVolume.net` program. The field is sent both to the first input tab of a `Slab` module and to an `Inquire` module, which works with a `Compute` and a `ForEachN` module to create a counter. The `Inquire` module counts the number of connections in eac dimension and, for the sample data set, returns [2,047 500]. The `Compute` module subtracts from the count in the *y* dimension to get '499', and the `ForEachN` module is then initialized t start counting at '0', incrementing by 1, until a count of '499' is reached. Each iteration is se to the third input tab of the `Slab` module.

For the first iteration, `ForEachN` sends the value '0', and the `Slab` module extracts the first item of each record from `NewCurtain.dx`. The extracted data are a 'slice' corresponding to the 'top' of the curtain representing the acoustic beam., and consists of 2,047 latitude/longitude values with a constant depth of –75 m, and the associated acoustic density values. The latitude/longitude values are computed as positions in space using three modules, `Mark`, `Compute`, and `UnMark`. The 'slice' of data is sent to the `Mark` module and 'marked,' which means that the data component is replaced with the positions component, after the original data component is stored in a temporary new component. The resultant 'marked' field is sent to a `Compute` module. The temporary replacement is necessary because `Compute` modules can only work on the data component of a field. The `Compute` module then calculates the data items as positions in space, and the `UnMark` module 'unmarks' the field, which means that the items are moved back into the positions component and the original data component is restored. The resultant field is sent to a `Remove` module, where the connections component is removed. The output of `Remove` is sent to an `Include` module where all the valid data are selected. The valid data at each depth is then sent to the first input tab of the `Regrid` module. The 40 x 40 grid (41 x 41 vertices) is constructed by using 1/40 of longitude and latitude ranges. The two-dimensional grid is the same for all depths. This process is completed for each iteration of the counter, until the count reaches '499' and `Slab` has extracted the last item of each record.

Part II

The `ShowBox` module receives the field `NewCurtain.dx` from the input box in Part I. This module creates a set of lines that define a boundary around the positions contained in the `NewCurtain.dx` field. The output of `ShowBox` is sent to an `Extract` module which extracts each dimension of the positions component and sends it to a `Compute` module. Each dimension is converted into one-dimensional vector. Each vector is sent to a `Statistics` module.

The first `Statistics` module extracts the maximum value in the *z* dimension (-75 m) of the boundary box and sends it to the first input tab ('a') of a `Compute` module. The second input tab ('b') of the `Compute` module receives its value from the `ForEachN` module in Part I, which is counting from '0' to '499'. The `Compute` module executes the expression *a - b* 0.5* for each iteration until the count reaches '499'. The output of `Compute` contains the values [-75.0, -75.5, -76.0, …., -324.5] and is used to set the vertical position (depth) of each two-dimensional interpolated grid.

The second and third `Statistics` modules extract the minimum and maximum values in the *x* dimension and *y* dimension of the boundary box, respectively, and send them to several `Compute` modules for processing.

The minimum values for both the *x* and *y* dimensions are received by a `Compute` module, which pairs them up, and sends them to the first input tab of a `Construct` module.

The first input tabs ('a') of two `Compute` modules receive the minimum *x* and *y* values, while the second input tabs ('b') receive the maximum *x* and *y* values. Both the *x* and the *y* `Compute` modules execute the expression *(b – a) / 40.* This means that the total distance along both the *x* and *y* dimensions of the boundary box is divided into 40 equal parts, so that there will be 40 squares in the *x-y* grid, with a distance of *(b – a) / 40* between each position. The output of the *x* and *y* `Compute` modules are paired with an additional `Compute` module, which sends the result both to the second input tab in the `Construct` module and to a further `Compute` module.

The `Construct` module creates a field of regular connections and regular positions using the information received from the *x* and *y* `Compute` modules. The first input tab specifies the origin of the grid (minimum *x* and *y* values). The second input tab sets delta, the distance between the positions. The third input tab is the number of positions in each dimension (set to 41, 41). The resultant field, which describes the grid, is sent to the second input tab of the `Regrid` module.

The final `Compute` module executes the expression *3*mag(a)* which calculates a value which is three times the magnitude of the distance between positions in the grid, and specifies the maximum distance from any point that a nearest neighbour will be considered in the interpolation. This value (0.0034003747 for the sample data set) is sent to the fourth input tab of the `Regrid` module.

Part III

The `Regrid` module maps scattered points onto a grid, using a number of nearest neighbours to assign each grid square a data value. Four input parameters have been set (Table 4). The output of `Regrid` is a grid with each vertex assigned an acoustic density value based on a nearest neighbour interpolation. Each time `ForEachN` executes, a new interpolated grid is created. Each grid is stored in memory until all 500 have been created. The `CollectSeries` module defines a template by which each grid is placed in a series with an associated position. The placement of each grid into the series occurs in the `Append/SetLocal/GetLocal` loop and the position of each grid is set by the `Compute` module in Section B (each position decreases by -0.5m, starting at –75m).

The `GetLocal` module works with the `SetLocal` module to place objects in and retrieve them from the cache. The first grid is stored in the cache by the `SetLocal` module while the second grid is created. A new grid is created, and then the cached grid is retrieved by the GetLocal module and appended onto the new grid with the `Append` module. Then the set of two grids is stored in memory with `SetLocal` while a third grid is being created, and the two grids are then retrieved by `GetLocal` and appended to the new third grid, and so on, until all

Table 4: Arguments for the `Regrid` module.

Input Number	Description	Value
1	A Field with positions to `Regrid` (scattered points)	A horizontal 'slice' of the acoustic beam (Section A)
2	A grid to use as a template on which to map the scattered points	A 40 by 40 grid (Section B)
3	The number of nearest neighbours	30
4	The maximum radius from a grid point that a nearest neighbour can be found	.0034003747 (Section B)

500 grids are created and appended. Once the counter reaches '499', the entire series (500 grids) is sent back to the `MakeVolume.net` program, where the series is 'stacked' in the *z* dimension and exported as `volume1.dx`.

5.8 The `MakeBathymetryNewGrid.net` program

The `MakeBathymetryNewGrid.net` program creates a bathymetric surface called `Bathymetry.dx` that incorporates data from the `NewBottomDepth.dx` file, raw bathymetry file, and `ShipTrack.dx` file (Fig. 24). Incorporating the bottom depth values from `NewBottomDepth.dx` with those from the bathymetry file ensures that the relevant acoustic data does not fall below the interpolated surface, as the bottom depth value for each ping in `NewBottomDepth.dx` is the actual depth at which the ping hits the bottom.

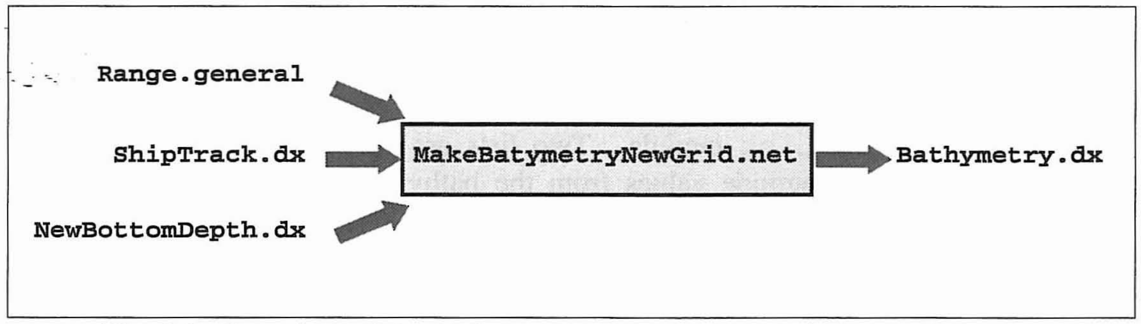


Fig. 24. Overview of the `MakeBathymetryNewGrid.net` program.

The `MakeBathymetryNewGrid.net` program has three sections (Appendix Fig. 11). In section A, the bathymetry file is processed and output as a field with a positional component containing latitude and longitude values and a depth component, with all the records corresponding to zero depths excluded. Section B creates a new field with a positions component containing the latitude and longitude values from the bathymetry file and `ShipTrack.dx`, and a data component containing the depth values from the bathymetry file and `NewBottomDepth.dx`. Section C outputs these values as three-dimensional vectors in a field called `Bathymetry.dx`, and creates an image of the bathymetric surface showing the triangular connections between each vector.

Section A.

An `Import` module imports the bathymetry file. The `Mark` module 'marks' the positions component (latitude, longitude, and depth values), and sends the resultant field to two `Compute` modules. The first `Compute` module declares the latitude and longitude values as two-dimensional vectors. The output is sent to the `UnMark` module which returns the latitude and longitude values to the positions component. The second `Compute` module declares the depth values as a scalar. A `Replace` module receives the output from the two `Compute` modules and places the scalar depth values in the data component of the field containing the two-dimensional longitude and latitude position vectors. The source field (first input tab) is the output from the second `Compute` and has a data component containing depth values, while the destination field (second input tab) is the output from `UnMark`, containing latitude and longitude values. `Replace` takes the data component of the source field and places it in the destination field, naming the new component 'depth'. The new field has depth values in the data component, and latitude and longitude values in the positions component. An Include module selects all depth values greater than or equal to 1.0, thereby excluding the incorrect zero depth values found in the data set.

Section B

The output from the Include module in Section A is sent to a Compute module where the depth values are made negative with the expression *-a*. The positions and data components are then extracted from the field using two Extract modules and sent to two List modules. Two Import modules import the fields ShipTrack.dx and NewBottomDepth.dx, which were created from the positional data file using the ValidShipTrack.net and MakeNewBottomDepth.net programs (Appendix Figs. 1 & 5). An Extract module extracts the positions component (normalized latitude and longitude values) from ShipTrack.dx and sends it to the first List module. A second Extract module extracts the data component (normalized depth values) from NewBottomDepth.dx and sends it to the second List module. Two lists are now created. One list (positions) contains the latitude and longitude values from the bathymetry file followed by those from ShipTrack.dx, while the other list (data) contains depth values from the bathymetry file followed by those from NewBottomDepth.dx. The positions list and the data list are sent to a Construct module, which combines them to form a new field, with a positions component from the positions list, and a data component from the data list. A Connect module is then used to create triangular connections between the positions.

Section C

The output from Connect is sent to a Mark module which 'marks' the positions component and sends the field to the first input tab of a Compute module. The second input tab receives the data component (depth values) from the output of Connect. These inputs are associated and defined as three-dimensional vectors with the Compute expression *a.x, a.y, b*, where *a* refers to the first input, and *b* refers to the second input. An UnMark module moves the output of Compute back to the positions component where it now consists of latitude, longitude, and depth (*x, y, z*) values. The field is exported with an Export module as bathymetry.dx. The triangular connections are made visible with the ShowConnections module. The scale of the image is adjusted to 1/1,000 in the *z* dimension using a Scale module with the expression *x, y, z = 1 1 0.001*. The bathymetry image is output using the Image module (Fig. 25).

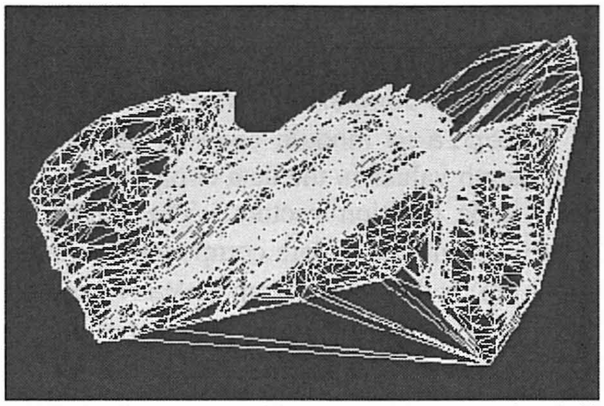


Fig. 25: Bathymetry with connections made visible.

5.9 The `CutVolume.net` program

The `CutVolume.net` program removes any data originating below the bottom from the three-dimensional image of the fish aggregation created by the `MakeVolume.net` program, so that the fish aggregations are only visible above the bathymetric surface (Fig. 26 and Appendix Fig. 12). Section A creates a new field called `Volume2.dx` which is the same as `Volume1.dx` but which has had the data which originate below the bathymetric surface removed for each position. Section B creates the new three-dimensional image of the fish aggregation.

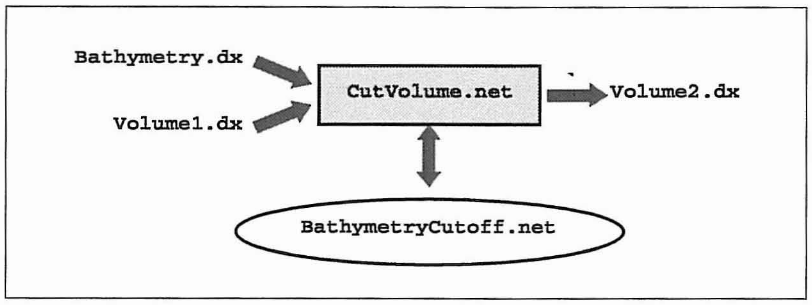


Fig. 26. Overview of the `CutVolume.net` program.

Section A

Two `Import` modules import the fields `Bathymetry.dx` and `Volume1.dx` and send them to the `BathymetryCutoff.net` macro which returns a field consisting of 500 grids, with latitude, longitude, and depth (x, y, z) values in the positions component, acoustic values in the data component, and an invalid positions component called `bathy_index` (Fig. 38). A `Stack` module stacks the grids in the z dimension (depth) and sends the resultant field to an `Include` module. The `Include` module selects all the data that has valid positions and sends the result to a `Mark` module, where the `bathy_index` component is marked. A second `Include` module selects all the data items where `bathy_index` has a value of 1, corresponding to valid depths, so that the depths below the bathymetric surface are excluded. The output is unmarked with an `UnMark` module. The output of `UnMark` is sent both to the `IsoSurface` module for further processing, and to an `Export` module where it is exported as `Volume2.dx`. The `IsoSurface` module uses its default setting, the data mean value, to create an isosurface, or interpolated density layer, and sends the result to a `Collect` module.

Section B

The connections in the `Bathymetry.dx` file are made visible with the `ShowConnections` module, and sent to the first input tab of a `Collect` module. The bathymetry is coloured grey and shaded using `Color` and `Shade` modules, and sent to the second input tab of the `Collect` module. The `Collect` module sends them as one group to a second `Collect` module, which also receives the output of the `IsoSurface` module in Section A. The isosurface (interpolated density layer) and the bathymetry are then displayed after being scaled in the z dimension using a `Scale` module (Fig. 27).

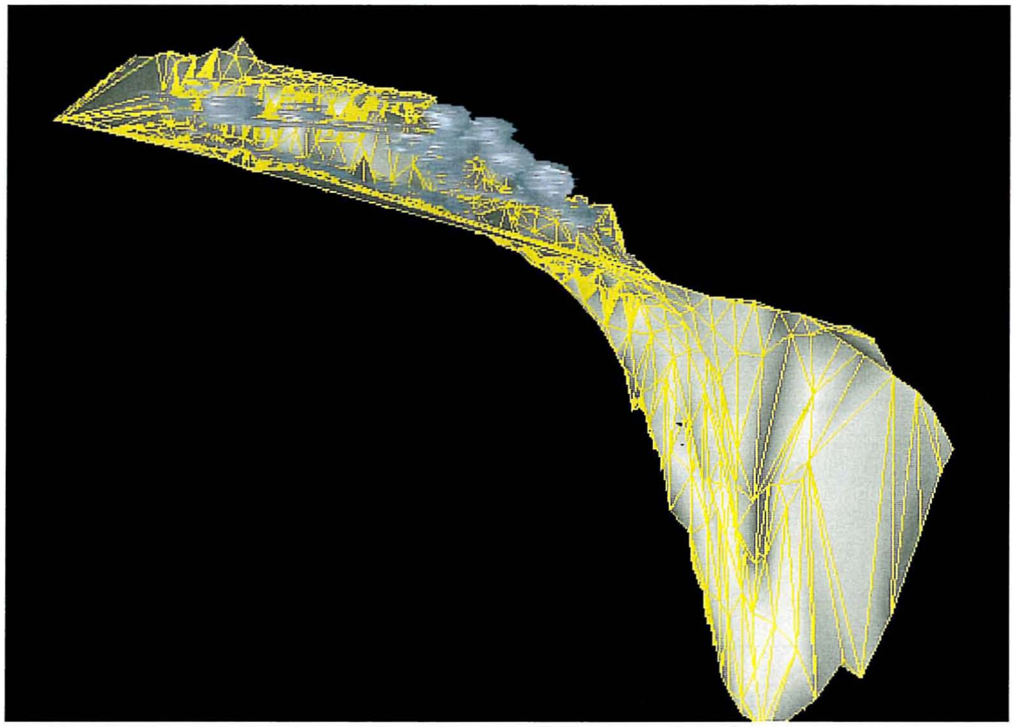


Fig. 27. An isosurface displayed on top of the bathymetry.

5.9.1 The `BathymetryCutoff.net` macro

The BathymetryCutoff.net macro has three parts (Appendix Fig. 13). In Part I, a field is created which contains positions extracted from `Volume1.dx` and depths interpolated from `Bathymetry.dx`.

Part I

Two `Input` modules receive the `Volume1.dx` and `Bathymetry.dx` fields from Section A of the `CutVolume.net` program. The `Volume1.dx` field is sent to a `Slice` module, while the `Bathymetry.dx` field is sent to a `Mark` module.

The `Slice` module drops the *z* dimension (depth) from the positions component of `Volume1.dx` and sends the resultant field to the first input tab('field to be mapped') of a `Map` module.

The `Mark` module marks the positions component of the `Bathymetry.dx` field, and sends the resultant field to two `Compute` modules. The first `Compute` module declares the latitude and longitude (*x* and *y*) values as two-dimensional vectors using the expression *a.x a.y*. The output is sent to the `UnMark` module which returns the latitude and longitude values to the positions component. The second `Compute` module declares the depth values as scalar using the expression *a.z*. A `Replace` module receives the output from the two `Compute` modules and places the scalar depth in the data component of the field containing the two-dimensional position vectors. The first input tab ('source') contains the output from the second `Compute` and has a data component containing depth values, while the second input tab ('destination') contains the output from UnMark, containing latitude and longitude values as positions. Replace takes the data component of the 'source' field and places it in the 'destination' field, naming the new component 'data'. The new field has depth values in the data component, and latitude and longitude values in the positions component. The output of Replace is sent to the second input tab ('map to be used') of the Map module.

The Map module now interpolates depth values for each position in the 'field to be mapped' using the positions and depths in the 'map to be used.' The resultant field therefore contains positions from the Volume1.dx file and depth values interpolated from the Bathymetry.dx file.

<u>Part II</u>

The Volume1.dx field from Section A of the CutVolume.net program is sent to both an Inquire module and to the first input tab of a Slab module. The Inquire module works with a Compute module and a ForEachN module to set up a counter to control 500 iterations ('0' to '499'). Each count is sent to the third input tab of the Slab module. The Slab module extracts each grid layer from Volume1.dx and sends the resultant fields to a Mark/Compute/UnMark/Replace group of modules which performs the same function as in Part I. For each grid a new field is created, and the latitude and longitude values (x, y) values are declared as two-dimensional vectors and placed in the positions component, while the depth (z) values are declared as scalar and placed in the data component of the field. For each grid layer that is processed, the depth (z) values will be a constant value, corresponding to the position of each layer in the z dimension.

<u>Part III</u>

The first input tab ('a') of a Compute module receives each output from the Replace module at the end of Part II, a field containing latitude/longitude positions and interpolated depths for each iteration of the ForEachN module. The second input tab ('b') of the Compute module receives the output of the Map module at the end of Part I, a field containing the positions of the two-dimensional grid and interpolated depths. Each field contains exactly the same positions and connections. The Compute module executes the expression *a>(b+1.0))?1:0*. This expression takes a depth from 'b', adds a value of '1.0', and asks whether or not this sum is less than the corresponding depth value from 'a'. If this is true, a value of '1' is returned; otherwise '0' is returned. This ensures that the height at which the volume is 'cut off' is always 1.0 unit higher then the interpolated bathymetry, so that the bathymetry is always below the 'cut off' depth. The list of '1's and '0's are placed in a new component named bathy_index and the entire field is sent to the second input tab of an Append module.

An Extract module extracts the data component of the scalar depth (z) values from the second Compute module in Part II. A Select module selects the first z and sends it to the third input tab of the Append module. For each grid layer that is processed, the depth (z) values will become a single value, corresponding to the position of each layer in the z dimension.Each time the ForEachN module in Part II executes, a new grid is created. Each grid is stored in memory until all 500 have been created, in the same way as in Part III of the

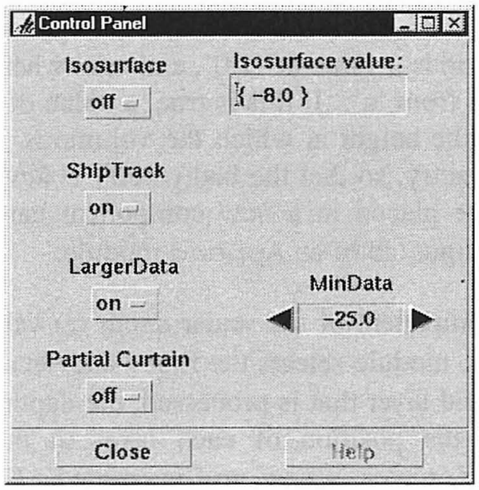
`LayerByLayerRegrid.net` macro. The `CollectSeries` module defines a template by which each grid is placed in a series with an associated position. The placement of each grid into the series occurs in the `Append/SetLocal/GetLocal` loop, and the position of each grid is defined by the *z* value in the third input tab of `Append`.

Once the `ForEachN` counter in Part II reaches ‘499’, the entire series (500 grids) is sent back to the `CutVolume.net` program, where the series is ‘stacked’ in the *z* dimension and exported as `volume2.dx`.

5.10 The `Iso.net` program

The `Iso.net` program organizes and displays the elements of the final image. This program has six ‘pages’ which correspond to the objects which are displayed in the final image. Pages are used in DX to compartmentalize sections of larger programs so that they are easier to follow. Each page is linked to other pages through transmitters and receivers. A transmitter assigns a certain name on one page and sends its object to a receiver with the same name on another page. This allows input and output tabs to be connected without an actual wire connection between modules.

In order to modify the final image, the user must change the parameters that are inputs for the Iso.net program. Interactors, which are contained in Control Panels, are interactive devices that can be used to manipulate the inputs to a visual program. Interactor stand-ins are used in the VPE to indicate which input to a module a given interactor is to control. There are six interactors that can be used to modify the final display of the widow rockfish visualization (Fig. 28).



Isosurface: off
Isosurface value: { -8.0 }
ShipTrack: on
LargerData: on
MinData: -25.0
Partial Curtain: off

Interactor	Description
Isosurface value	Used to specify the value(s) for any isosurface(s) to be displayed
Isosurface	Used to turn on/off the isosurface(s) set under ‘Isosurface value’ parameter
ShipTrack	Used to turn on/off a line that depicts the ship track
MinData	Used to set a minimum value (log scale) for data to be displayed on the ‘curtain’
LargerData	Used to turn on/off the data values above the ‘MinData’ setting
Partial Curtain	Used to turn on/off a partial transparent curtain

Fig. 28. A control panel containing six interactors which can modify the display of an image.

5.10.1 The Iso page

The Iso page allows the user to create one or many isosurfaces which give the interpolated acoustic densities for selected depths. These then are displayed in the final image (Fig. 42). An `Import` module imports `Volume2.dx` and sends it to an `Include` module where all the data are selected. The output of `Include` is sent to a `Compute` module which computes a log scale for the data and sends the output to an `Isosurface` module and to a `ScalarList` module. The iso-value to be used for computing the isosurface is set by the user in the `ScalarList` control panel, where one or a range of values can be entered. The isosurface is coloured using the `ColorMapUsed` receiver, which received its input from the corresponding transmitter on the Curtain Data page. After colouring, the isosurface is sent to a `Switch` module which determines whether the isosurface information will be passed through the `IsoSurface_3D` transmitter to the image page. The user can control the `Switch` module by turning the isosurface on or off in the Selector module. If the isosurface is turned on, it is one of the objects which is received and collected for the final image on the Imaging page.

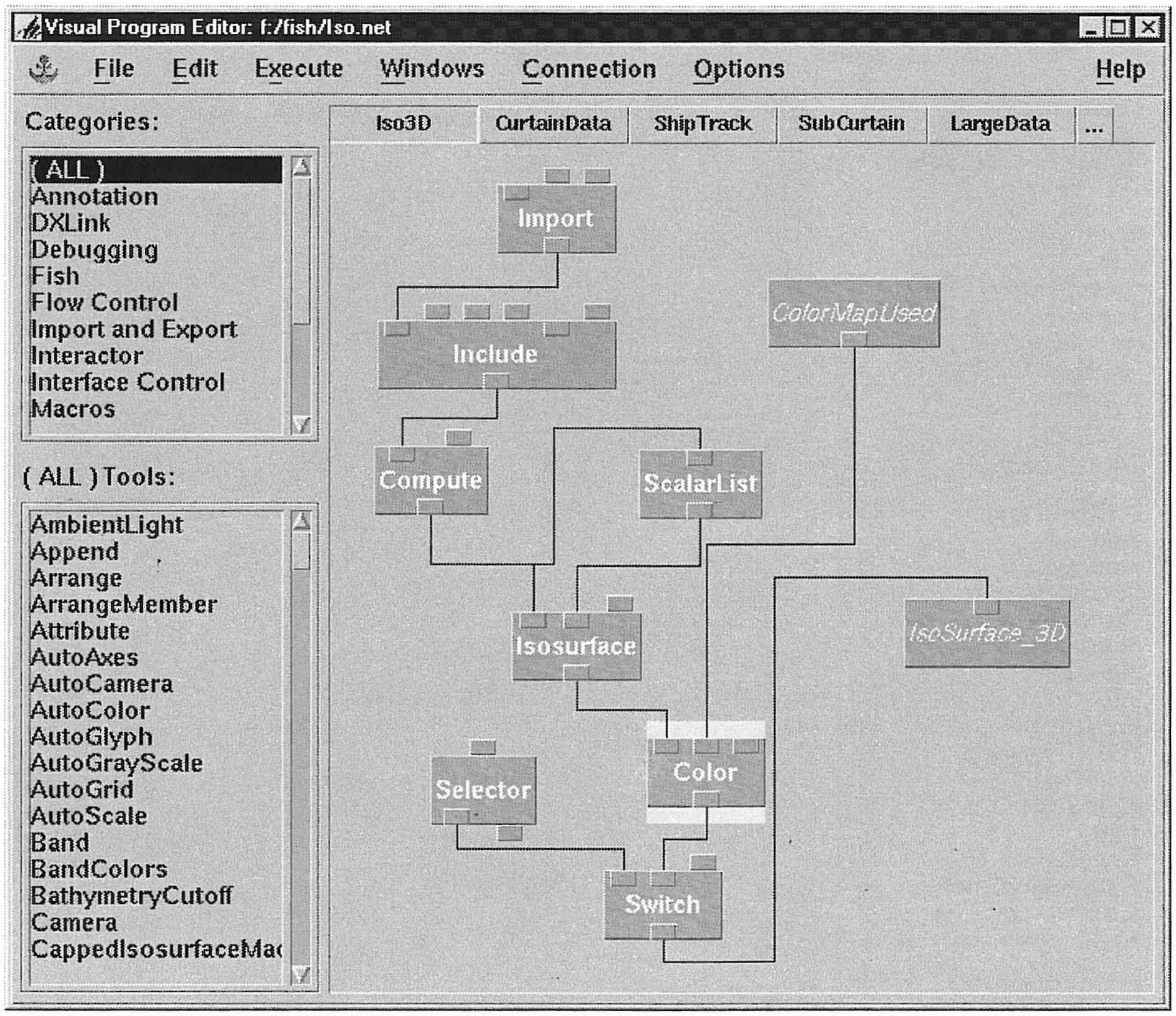


Fig. 29. The Iso page of the `Iso.net` program.

5.10.2 The Curtain Data page

The Curtain Data page allows the user to choose the colour scheme for the final image, and also converts the data from `NewCurtain.dx` to a logarithmic scale for use by the ShipTrack page (Fig. 30). An `Import` module imports `NewCurtain.dx` and sends it to a `Compute` module which applies a logarithmic scale. The output of `Compute` is sent to the `LogCurtainData` transmitter and to a `ColorMap` module. The `ColorMap` module allows the user to define the colour map (colour scheme) to be applied to the data. This map is sent to the `ColorMapUsed` transmitter and to a `ColorBar` module, which sets parameters for the placement the legend in the final image. The output of the `ColorBar` module is sent to the `ColorCaption` transmitter, which is received and collected for the final image on the Imaging page.

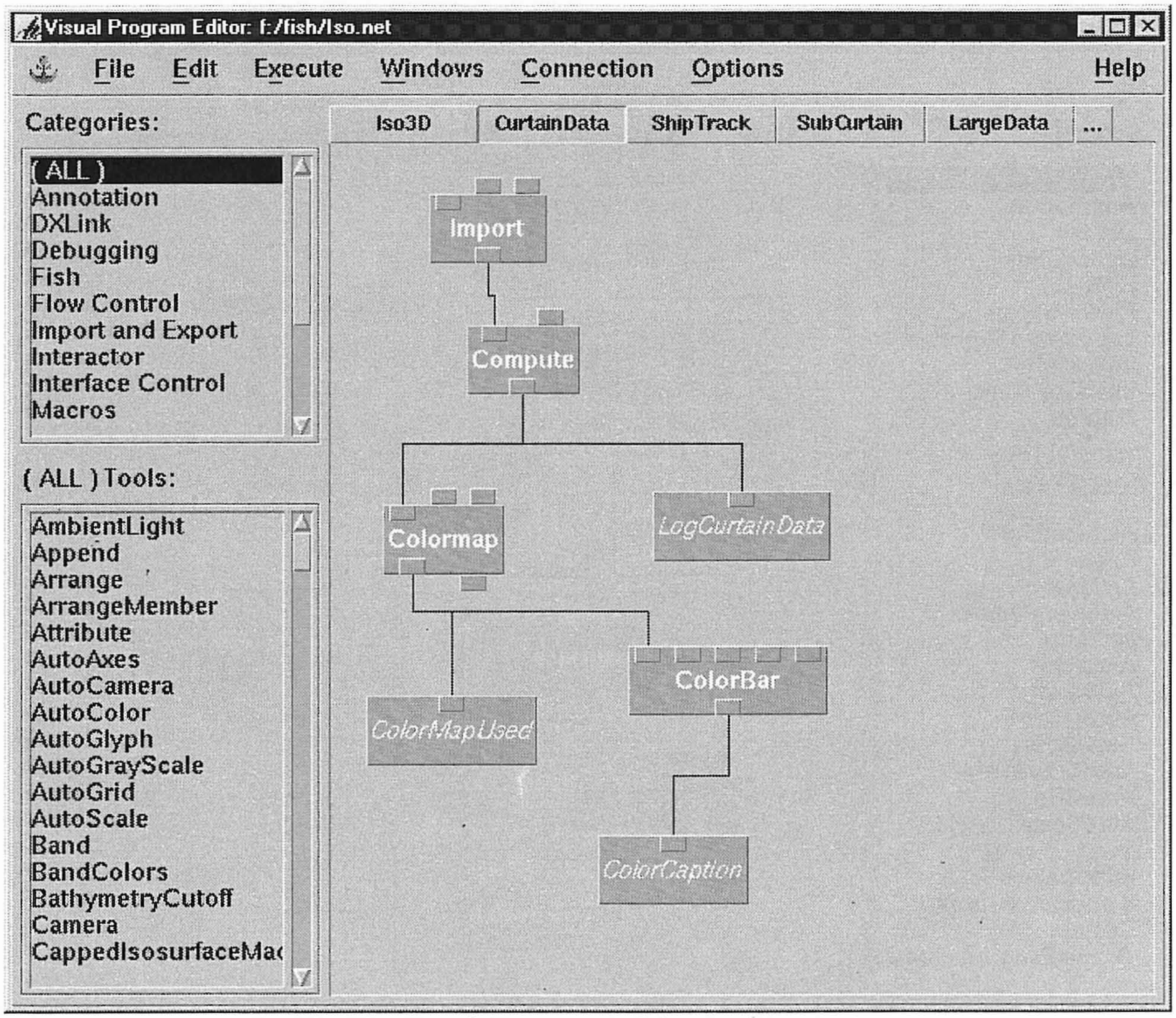


Fig. 30. The CurtainData page of the `Iso.net` program.

5.10.3 The Ship Track page

The Ship Track page allows the user to display the surface track of the acoustic vessel along the transects (Fig. 31). Data from `NewCurtain.dx`, which has been converted to a logarithmic scale, is received from the Curtain Data page by the LogCurtainData receiver. The data are sent to a `Slab` module, which extracts a 'slice' from the top of the 'curtain' of data. The connections of the extracted slice of data are displayed using the `ShowConnections` module and coloured white with the `Color` module. The output from the `Color` module is sent to a `Switch` module which determines whether the ship track will be passed through the `ShipTrack` transmitter to the Image page. The user can control the `Switch` module by turning the ship track on or off in the `Selector` module. If the ship track is turned on, it is one of the objects which is received and collected for the final image on the Imaging page.

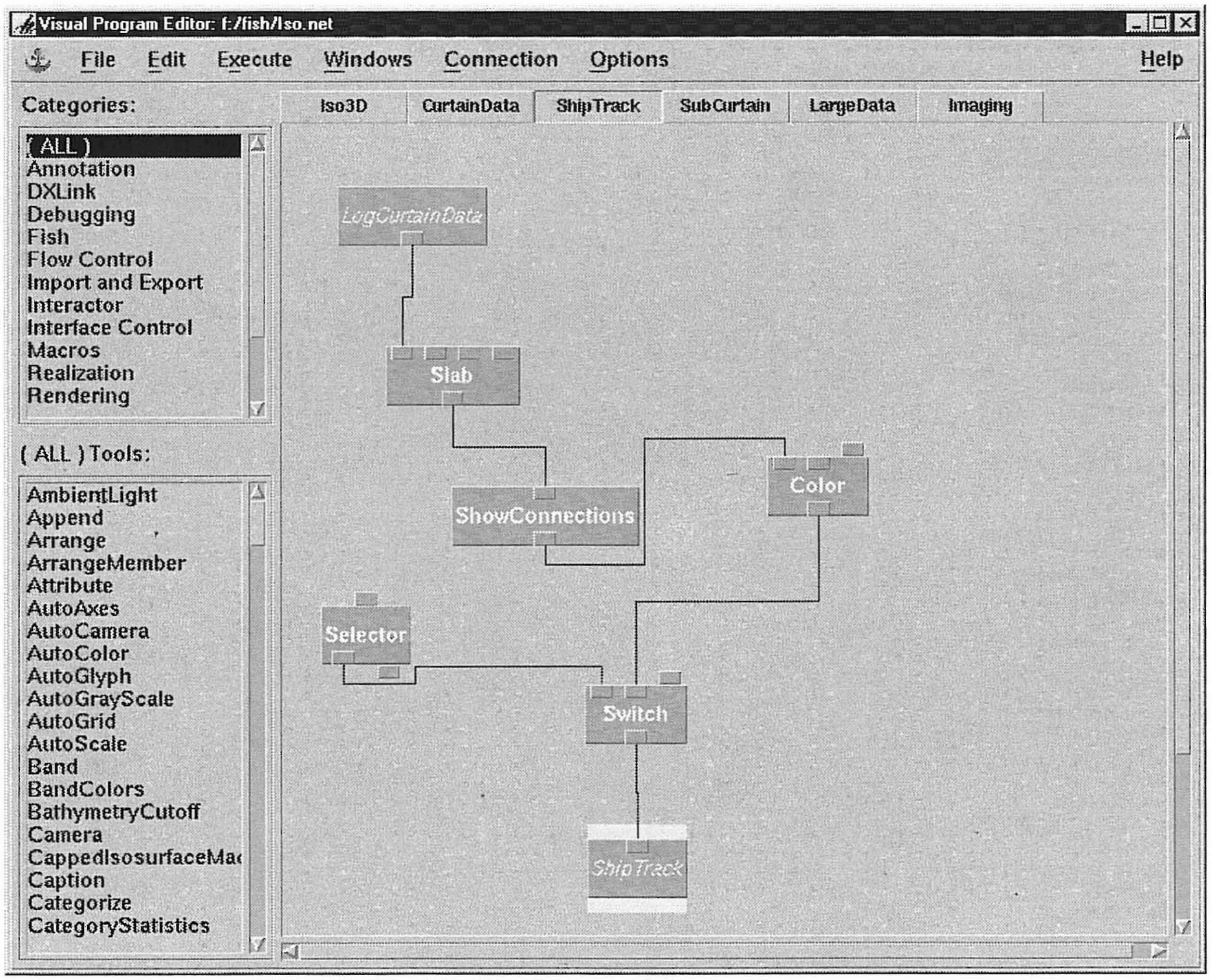


Fig. 31. The ShipTrack page of the `Iso.net` program.

5.10.4 The SubCurtain page

The SubCurtain page allows the user to create a 'partial curtain' of data, incorporating the last 400 positions from the `NewCurtain.dx` file (Fig. 32). Data from `NewCurtain.dx`, which has been converted to a logarithmic scale, is received from the Curtain Data page by the `LogCurtainData` receiver. The data are sent to a `Slab` module, which extracts a 'slab' corresponding to the last 400 of the 500 positions on the 'curtain.' This partial curtain is sent to a `Color` module where it is coloured with the colour scheme from the `ColorMapUsed` receiver. The output of `Color` is passed to a `Switch` module which determines whether the partial curtain will be passed through the `SubCurtain` transmitter to the Imaging page. The user can control the `Switch` module by turning the partial curtain on or off in the `Selector` module. If the partial curtain is turned on, it is one of the objects which is received and collected for the final image on the Imaging page.

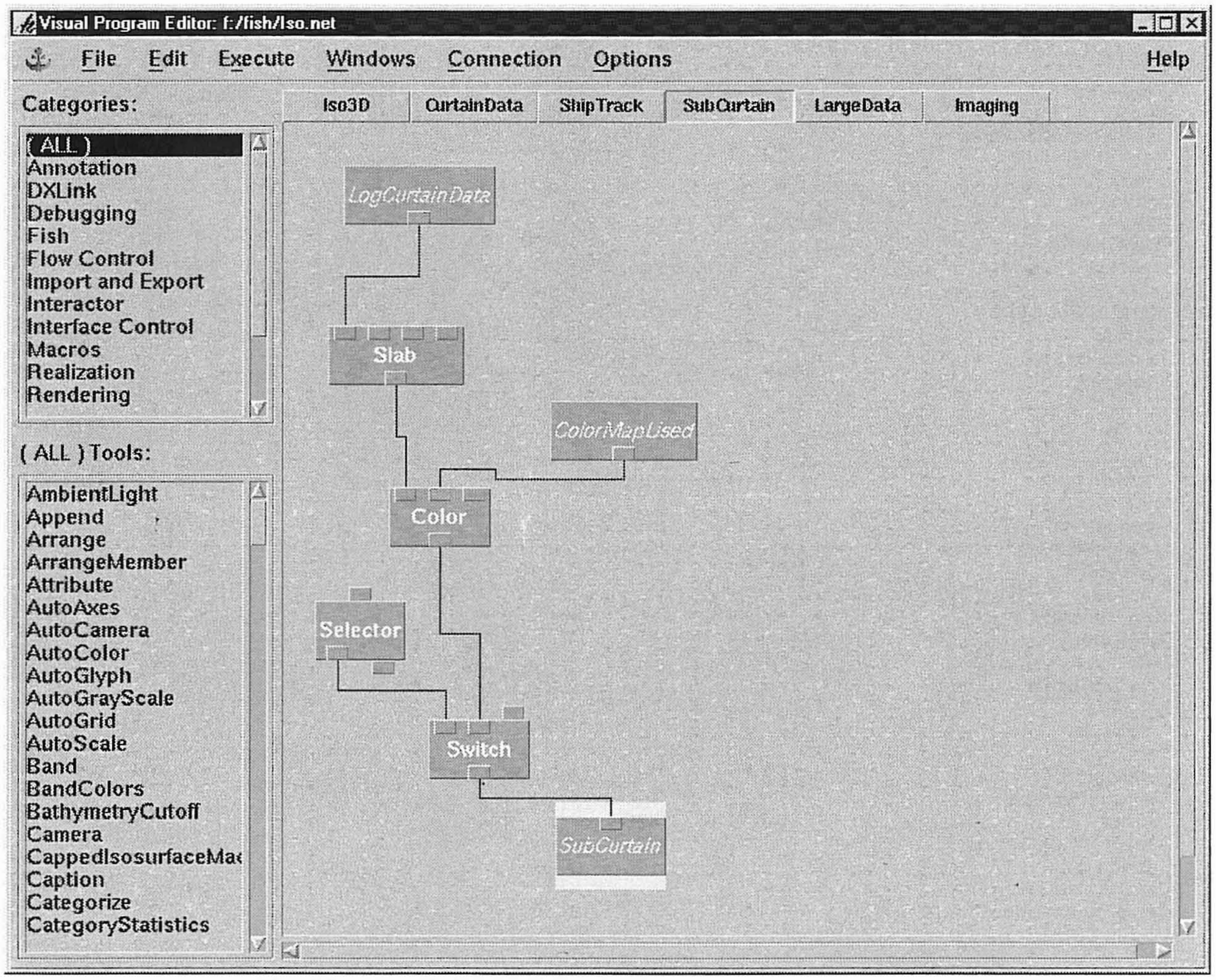


Fig. 32. The SubCurtain page of the `Iso.net` program.

5.10.5 The Large Data page

The Large Data page allows the user to set a minimum value for the data from `NewCurtain.dx` to be included in the final image (Fig. 33). Data from `NewCurtain.dx`, which has been converted to a logarithmic scale, is received from the Curtain Data page by the `LogCurtainData` receiver. The data are sent to the first input tab of an `Include` module. The second input tab of this module defines the minimum data to be used for display. This value is set by the user with the MinData option in the `ScalarList` control panel. The output from `Include` is sent to a `Color` module where it is coloured using the colour scheme in the `ColorMapUsed` receiver. The output of `Color` is passed to a `Switch` module which determines whether it will be passed through the `LargeData` transmitter to the Imaging page. The user can control the `Switch` module by turning the large data on or off in the `Selector` module. If the large data option is turned on, this object is received and collected for the final image on the Imaging page.

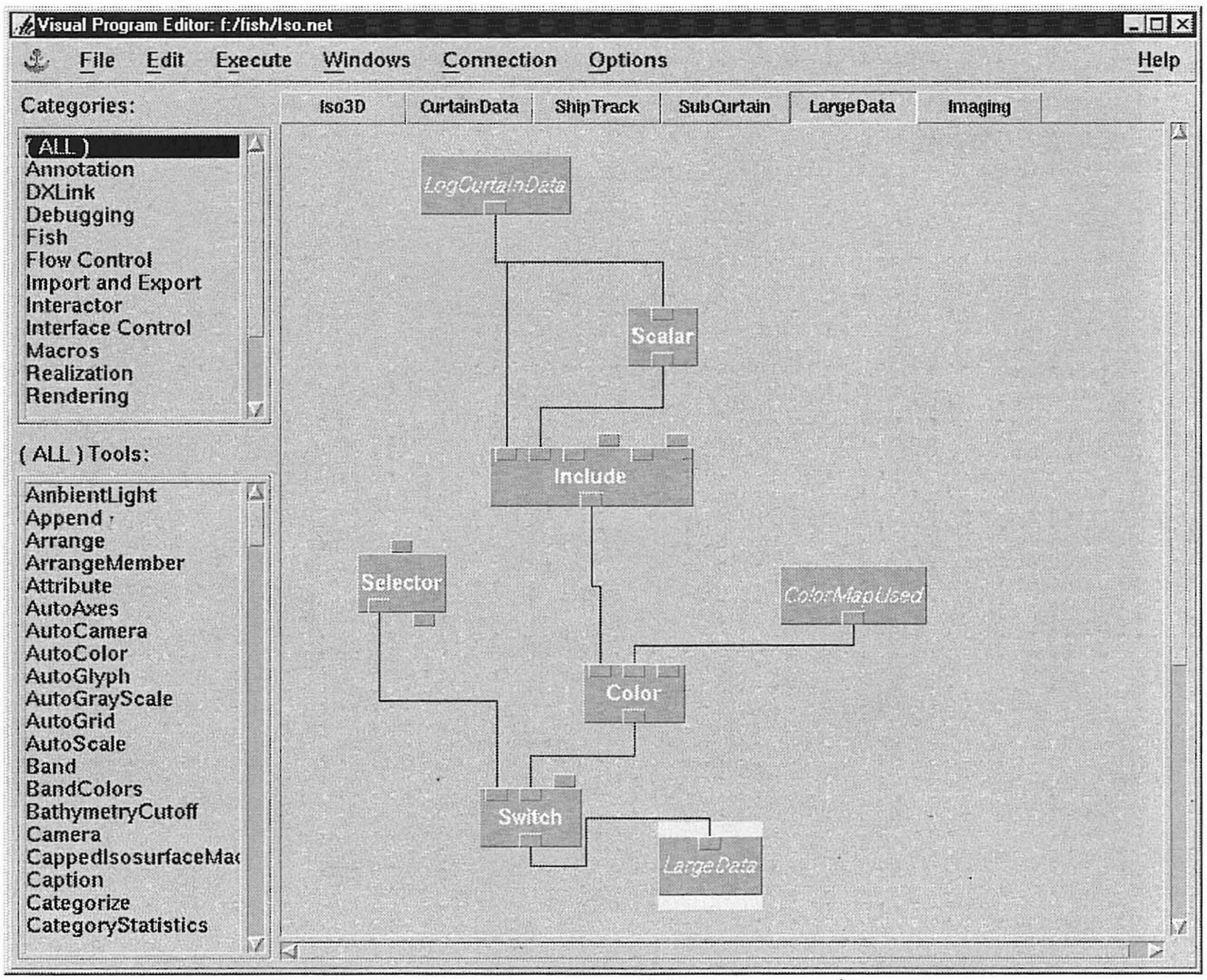


Fig. 33. The LargeData page of the `Iso.net` program.

5.10.6 The Imaging page

The Imaging page displays the final image (Fig. 34). A `Collect` module gathers all of the objects sent by the five transmitters on previous pages in five corresponding receivers. In addition, an `Import` module imports the `bathymetry.dx` file. The bathymetry data are coloured grey and shaded using the `Color` and `Shade` modules, and sent to the `Collect` module. The output of `Collect` is scaled in the z dimension using a `Scale` module before being sent to `Image` and displayed.

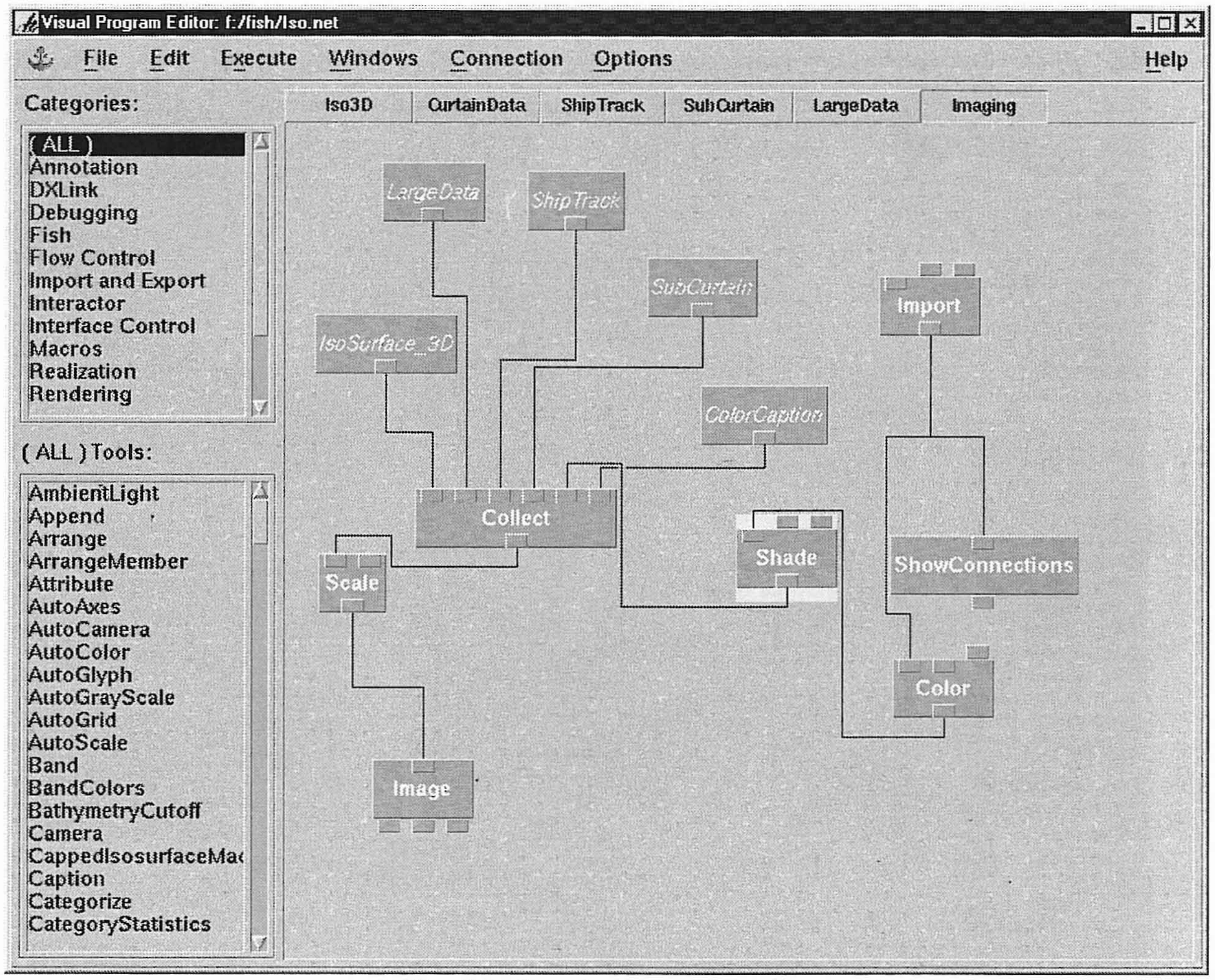


Fig. 34: Imaging page of Iso.net

6.0 THE IMAGE

The result of the visual programs is an image displaying the location and distribution of widow rockfish along with the bathymetric surface of the survey area. Some image parameters and a corresponding example figure are shown in Table 5 and Fig. 35.

There are a number of view options available in a DX Image window. The user can pan and zoom around the object, rotate the image, set view angles and choose different projections (e.g. orthographic) for image display. The image can also be saved in a number of different formats.

The parameters in `ScalarList` can be continually adjusted to fine-tune the visualization requirements of the user. The colours for the bathymetric surface and the colour bar legend can also be adjusted to achieve the most effective display.

An example of one of the final images produced for an entire micro-survey is shown in Fig. 36. The `ColorMap` contained a maximum value of –7.0 and a minimum value of –14.5. The MinData value in the control panel was set to –24, and only the LargeData parameter was turned on. An additional page was added to Iso.net to display the track of the vessel as a white line on the ocean bottom (Fig. 36). The bathymetry was also shaded with a different colour. Various rotations were tested on the image so that the relationship of fish distribution and surface features could be discerned.

Table 5. Settings in the control panel to create Fig. 36.

Interactor	Setting	Description
Isosurface value	Off	Used to specify the value(s) for any isosurface(s) to be displayed
Isosurface	Nil	Used to turn on/off the isosurface(s) set under 'Isosurface value' parameter
ShipTrack	On	Used to turn on/off a line that depicts the ship track
MinData	-24.0	Used to set a minimum value (log scale) for data to be displayed on the 'curtain'
LargerData	On	Used to turn on/off the data values above the 'MinData' setting
Partial Curtain	Off	Used to turn on/off a partial transparent curtain

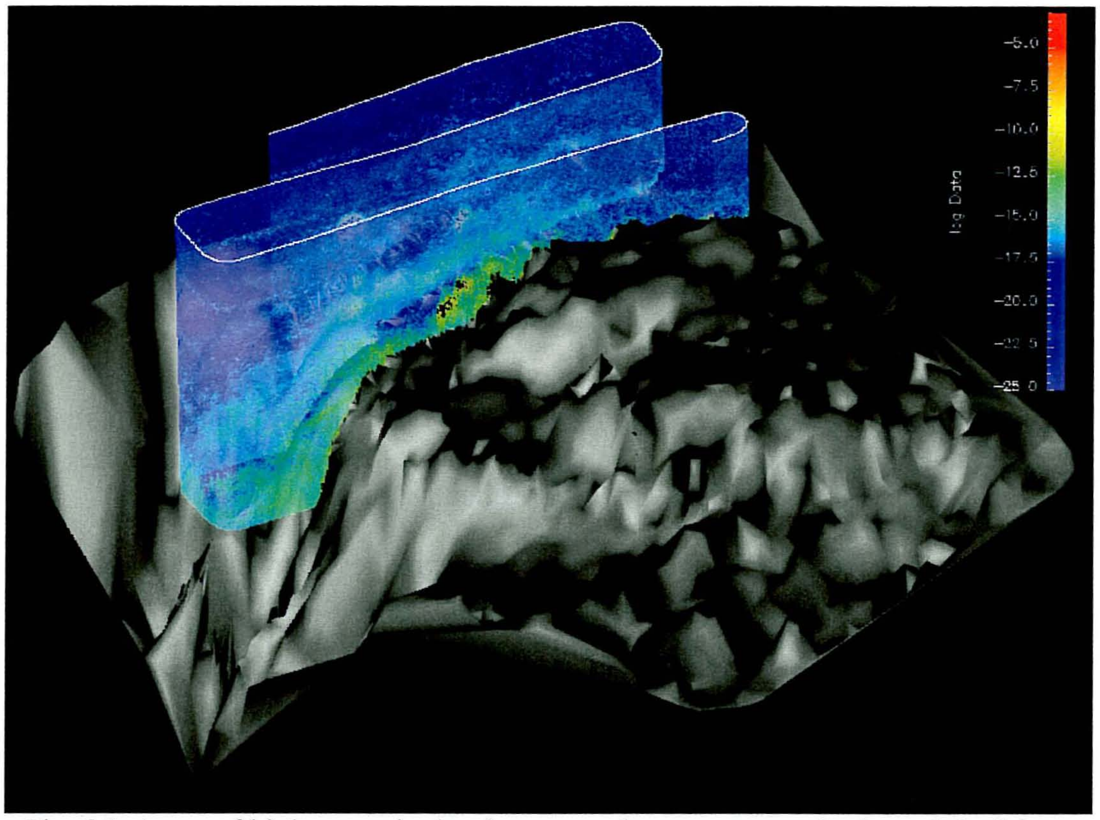


Fig. 35: Areas of high acoustic density are made apparent on the 'curtain' of data.

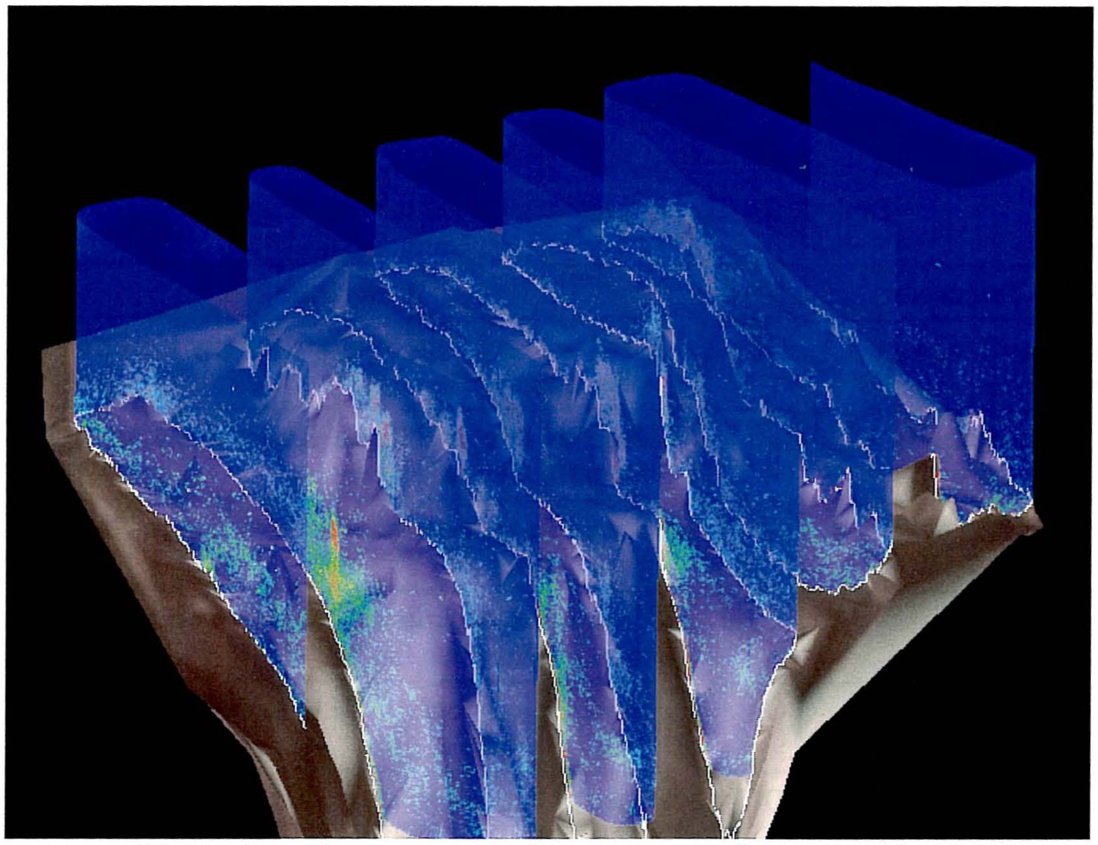


Figure 36. Further enhancements made to an image of an entire micro-survey.

7.0 PROCESSING OF ADDITIONAL DATA SETS

Other acoustic data sets may be processed, provided they are organized in the form of positional, acoustic, and bathymetric data files, in the same manner as the data for this study. The steps required for the processing of additional data sets are outlined below.

1) Create `.general` files for the positional, acoustics and bathymetric data files.
    - Importing the data into a spreadsheet is a fast method of extracting the number of records in the data files, plus the number of lines in the header.

2) Run `ValidShipTrack.net`
    - Double-click on the `Import` module and type in the full path for the location of the `.general` file name for the positional data.
    - Double click on the `Export` module and type in the full path for the resultant `*.dx` file.
    - Choose `Execute Once` from the Execute Menu.

3) Run `MakeBottomDepth.net`
    - Double-click on the `Import` module and type in the full path for the location of the `.general` file name for the positional data.
    - Double click on the `Export` module and type in the full path for the resultant `*.dx` file.
    - Choose `Execute Once` from the Execute Menu.

4) Run `MakeNewBottomDepth.net`
    - Double-click on the `Import` module and type in the full path for the location of the `*.dx` file created in step 3.
    - Double click on the `Export` module and type in the full path for the resultant `*.dx` file.
    - Choose `Execute Once` from the Execute Menu.

5) Run `ShipTrack3D.net`
    - Double-click on the `Import` module and type in the full path for the location of the `*.dx` file created in step 1.
    - Double click on the `Export` module and type in the full path for the resultant `*.dx` file.
    - Choose `Execute Once` from the Execute Menu.

6) Run `MakeCurtain.net`
    - Double-click on the first `Import` module and type in the full path for the location of the `ShipCurtain.dx` file. Note that this file must be edited for EACH data set by changing the number of items to reflect the number of records in the data set (Fig. 37). If a data set contains different values for pelagic upper, pelagic lower, and pelagic count, these values must also be reflected in the `ShipCurtain.dx` file, by setting the origin (pelagic upper) and delta (depth interval for each consecutive reading).

Fig. 37. Edits made to a ShipCurtain.dx file that contained 3412 records. Pelagic upper = 75, delta = 0.5 m, pelagic lower =325, and pelagic count = 500.

```
object 1 class array type float rank 1 shape 3 items 3412 data file ShipTrack3D
object 2 class  regulararray count 325
origin 0.0 0.0 -75.0
delta 0.0 0.0 -0.5
object 3 productarray
     term 1
     term 2
object 4 class gridconnections counts 3412 500

object 'field' class field
component 'positions' 3
component ' connections' 4
end
```

- Double-click on the second Import module and type in the full path for the location of the .general file name for the acoustics data.
- Double click on the Export module and type in the full path for the resultant *.dx file.
- Choose Execute Once from the Execute Menu.

7) Run MakeNewCurtain.net
    - Double-click on the first Import module and type in the full path for the location of the *.dx file created in step 6.
    - Double-click on the second Import module and type in the full path for the location of the *.dx file created in step 4.
    - Double click on the Export module and type in the full path for the resultant *.dx file.
    - Choose Execute Once from the Execute Menu.

8) Run MakeVolume.net
    - Double-click on the Import module and type in the full path for the location of the *.dx file created in step 7.
    - Double click on the Export module and type in the full path for the resultant *.dx file.
    - Choose Execute Once from the Execute Menu.

9) Run MakeBathymetryNewGrid.net
    - Double-click on the left Import module and type in the full path for the location of the .general file for the bathymetry data.
    - Double-click on the middle Import module and type in the full path for the location of the *.dx file created in step 1.
    - Double-click on the right Import module and type in the full path for the location of the *.dx file created in step 4.
    - Double click on the Export module and type in the path for the resultant *.dx file.
    - Choose Execute Once from the Execute Menu.

10) Run `CutVolume.net`

- Double-click on the first `Import` module and type in the full path for the location of the `*.dx` file created in step 9.
- Double-click on the right `Import` module and type in the full path for the location of the `*.dx` file created in step 8.
- Double click on the `Export` module and type in the full path for the resultant `*.dx` file.
- Choose `Execute Once` from the Execute Menu.

11) Run `Iso.net`

- Click on the Curtain Data page.
- Double-click on the `Import` module and type in the full path for the location of the `*.dx` file created in step 7
- Double-click on `ColorMap` and type in the minimum and maximum values for the colour bar.
- Click on the Iso page.
- Double-click on the `Import` module and type in the full path for the location of the `*.dx` file created in step 9.
- Set the parameters in the `ScalarList` control panel.
- Click on the Imaging page.
- Double-click on the `Import` module and type in the full path for the location of the `*.dx` file created in step 8.
- Choose `Execute Once` from the Execute Menu.
- Adjust `ScalarList` parameters, bathymetry and colour bar colours to meet final image requirements.

8.0 RESULTS

Using the IBM Visualization Data Explorer, ten visual programs and four macros were developed in order to display widow rockfish acoustic data in three dimensions. The data provided from the widow rockfish survey were in the form of three ASCII files containing acoustic, geographical position, and bathymetry data. The programs processed data from these three files to create the different components of the final image. Processing involved selecting and combining data from the three files and converting it to a form which could be displayed in three dimensions. Position data were normalized by removing any duplicate locations which arose due to recording errors. New values were interpolated from the neighbouring unique locations to replace the duplicates. Acoustic data that originated below the bottom were removed to improve the clarity of the final image. Interpreted three dimensional data sets and images were created by stacking interpolated two dimensional layers from each 0.5 m depth interval. Interpolated data values were based on a weighted nearest neighbour average that considered all known values within a specified radius.

The main components of the final image are a three-dimensional interpolated bathymetric surface, a surface line that gives the vessel track, and a two-dimensional 'curtain' of non-interpolated acoustic data that shows a cross-section through the fish concentrations in the water column. We also generated three-dimensional displays of interpolated acoustic density data that would render the two dimensional curtain image as three dimensional aggregations of fish. A control panel with several interactors allows adjustment of the final image. The user can control the contents and colour scheme, and can directly interact with the objects in the DX Image window using mouse-driven rotation and zoom tools.

Any acoustic data set containing backscatter cross-section, position, and depth in ASCII files can be used to create three-dimensional images using these programs. The flexibility of the DX visualization process means that three-dimensional visualization can be used in a wide range of acoustic applications. Any feature of the ocean that can be detected using acoustic methods could be displayed as a three-dimensional image using these programs. In particular, acoustic data pertaining to fish or plankton distribution and abundance could be compared visually over time and between different locations, providing an intuitive visual tool to accompany rigorous estimation methods. Some examples of the application of three dimensional acoustic visualization to planktonic research can be found in Greene et al. (1994) and Greene et al. (1998).

For the widow rockfish survey, the three-dimensional images played a significant role in the analysis of rockfish biomass, distribution, and diel behaviour. They also proved to be an invaluable tool in facilitating the dialogue between industry representatives and biologists.

9.0 LITERATURE CITED

Greene C.H., P.H. Wiebe, and J.E. Zamon. 1994. Acoustic Visualization of Patch Dynamics in Ocean Ecosystems. Oceanography 7(1): 4-12.

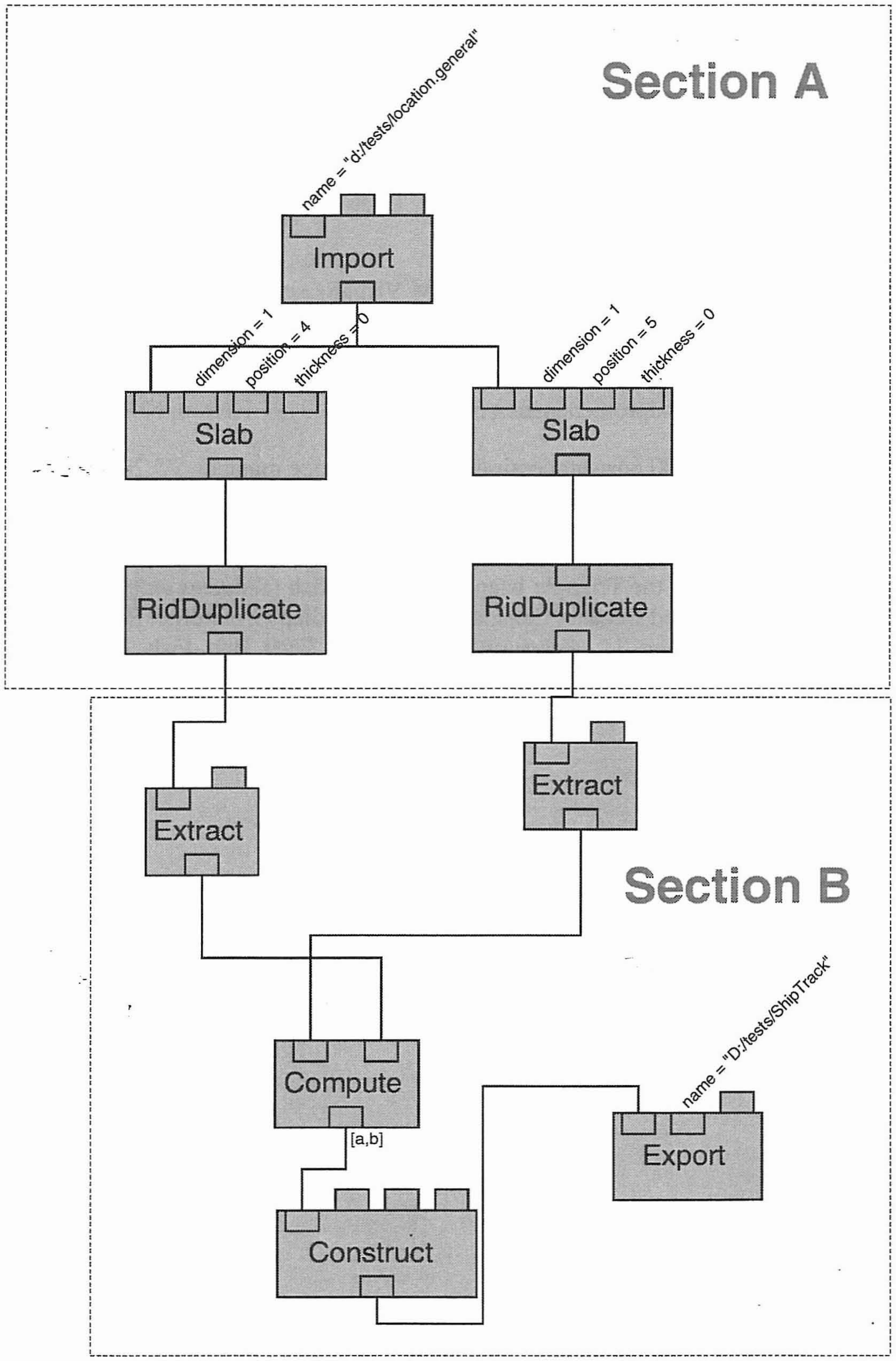
Greene C.H., P.H. Wiebe, C. Pelkie, J.M. Popp, and M.C. Benfield. 1998. Three-Dimensional Acoustic Visualization of Zooplankton Patchiness. Deep Sea Research II Special Issue on Biological Oceanography.

International Business Machines Corporation. 1997. IBM Visualiazation Data Explorer Users' Guide Version 3, Release 1, Modification 4.

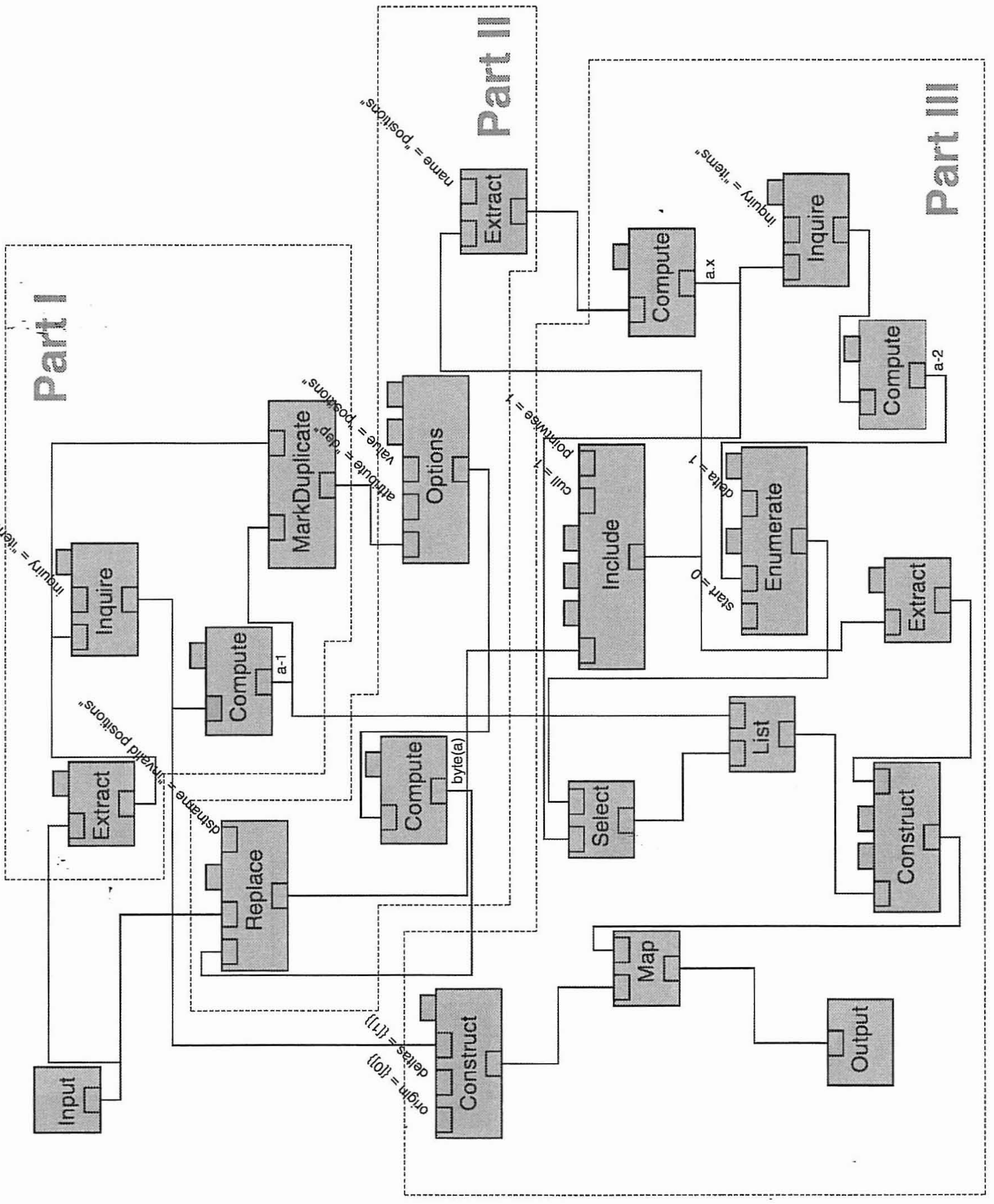
SIMRAD. 1993a. SIMRAD EK500 scientific echo sounder reference manuals V4.01. SIMRAD Subsea A/S, Standpromenenaden 50, Box 111, N-3191 Horten, Norway.

SIMRAD. 1993b. SIMRAD BI500 post-processing system reference manuals V5.20. SIMRAD Subsea A/S, Standpromenenaden 50, Box 111, N-3191 Horten, Norway.

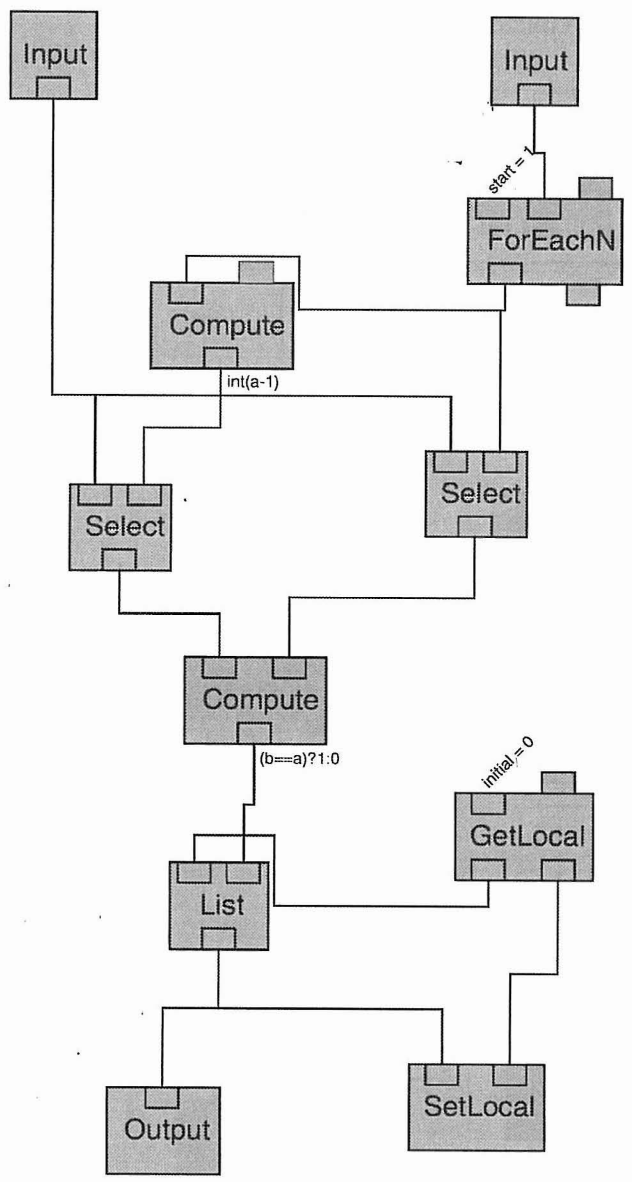
Stanley, R.D., A.M. Cornthwaite, R. Kieser, K. Cooke, G.D. Workman, and B. Mose. 1999. An acoustic biomass survey of the Triangle Island widow rockfish (*Sebastes entomelas*) aggregation by Fisheries and Oceans, Canada and the Canadian Groundfish Research and Conservation Society, January 16 - February 7, 1998. Can. Tech. Rep. Fish. Aquat. Sci. 2262: 51 p.



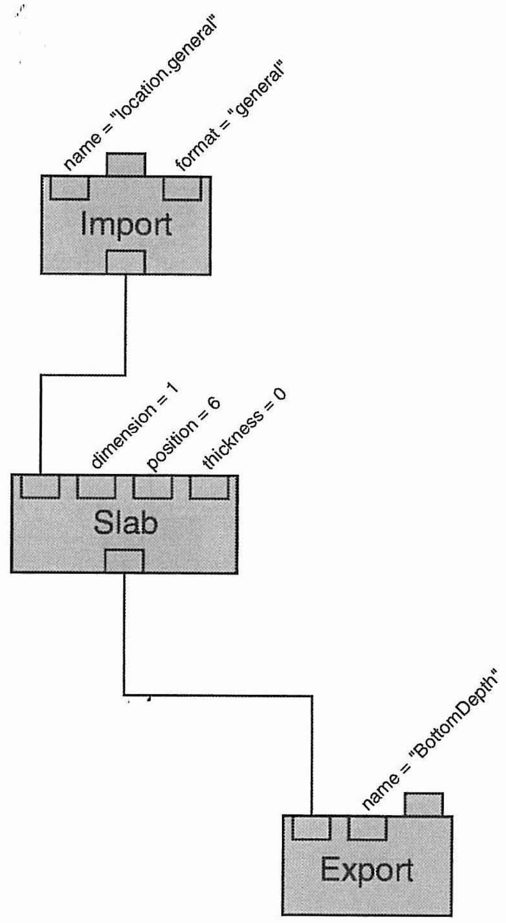
Appendix Fig. 1. The `ValidShipTrack.net` program.



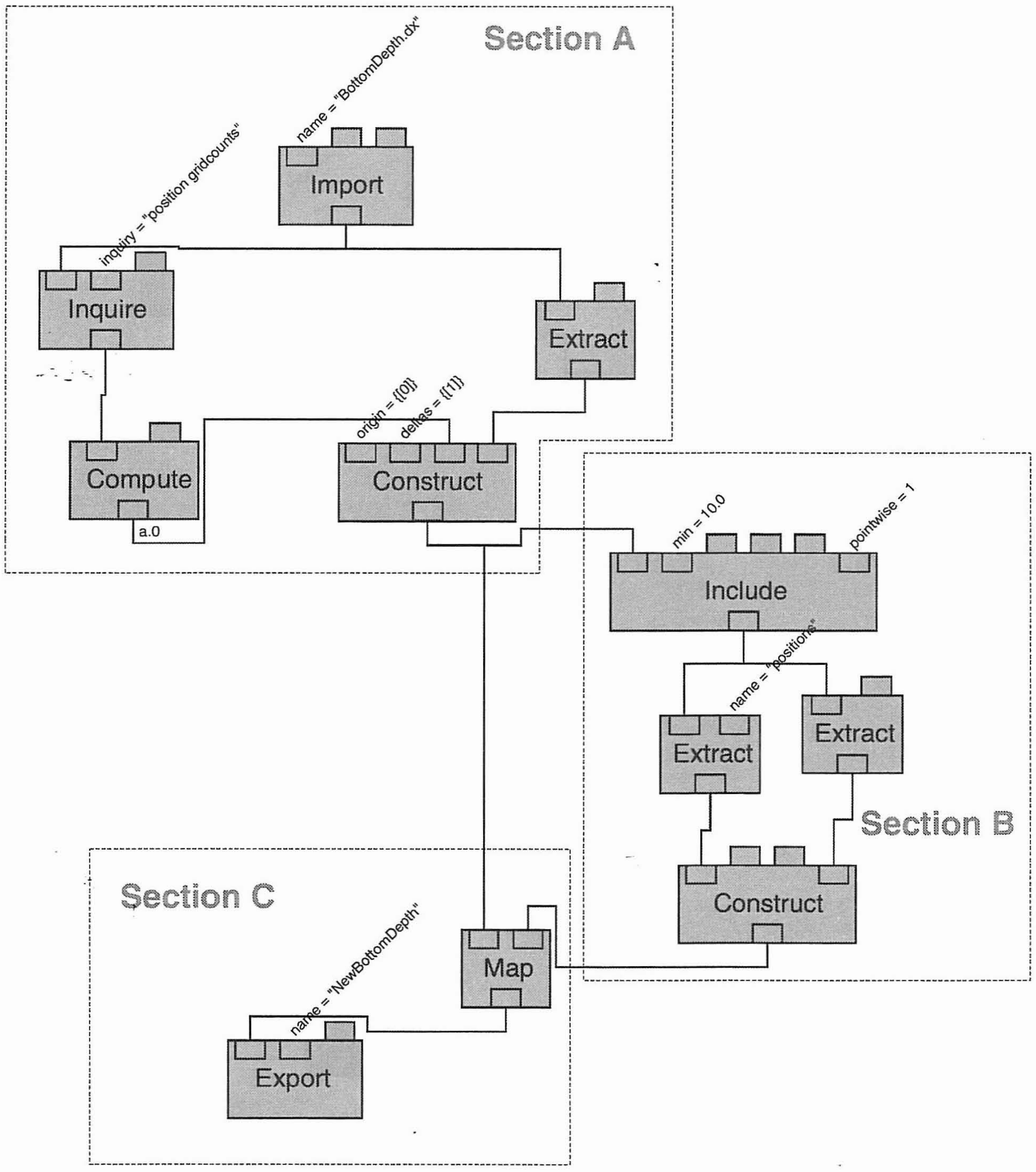
Appendix Fig. 2. The `RidDuplicate.net` macro.



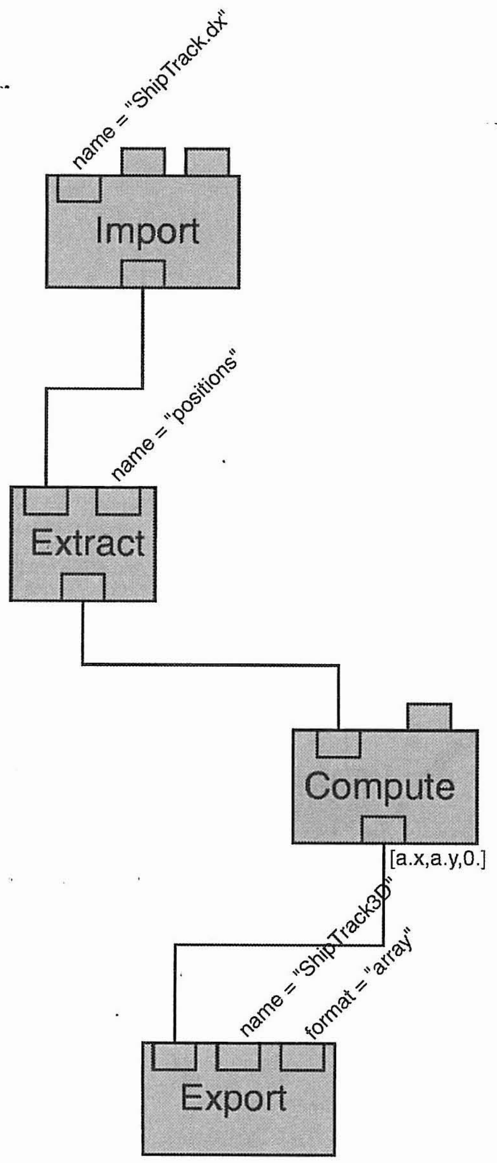
Appendix Fig. 3. The `MarkDuplicate.net` macro.



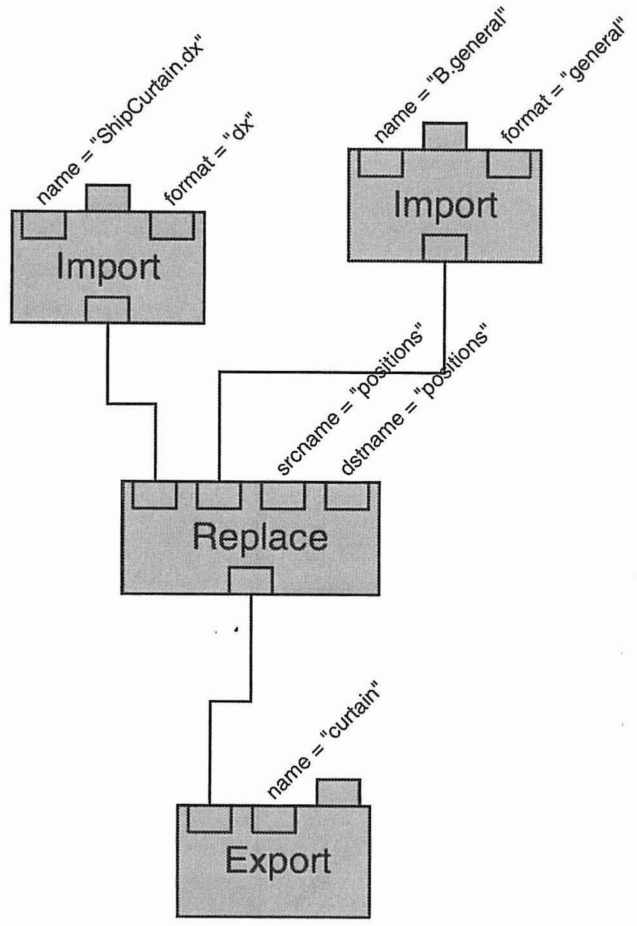
Appendix Fig. 4. The `MakeBottomDepth.net` program.



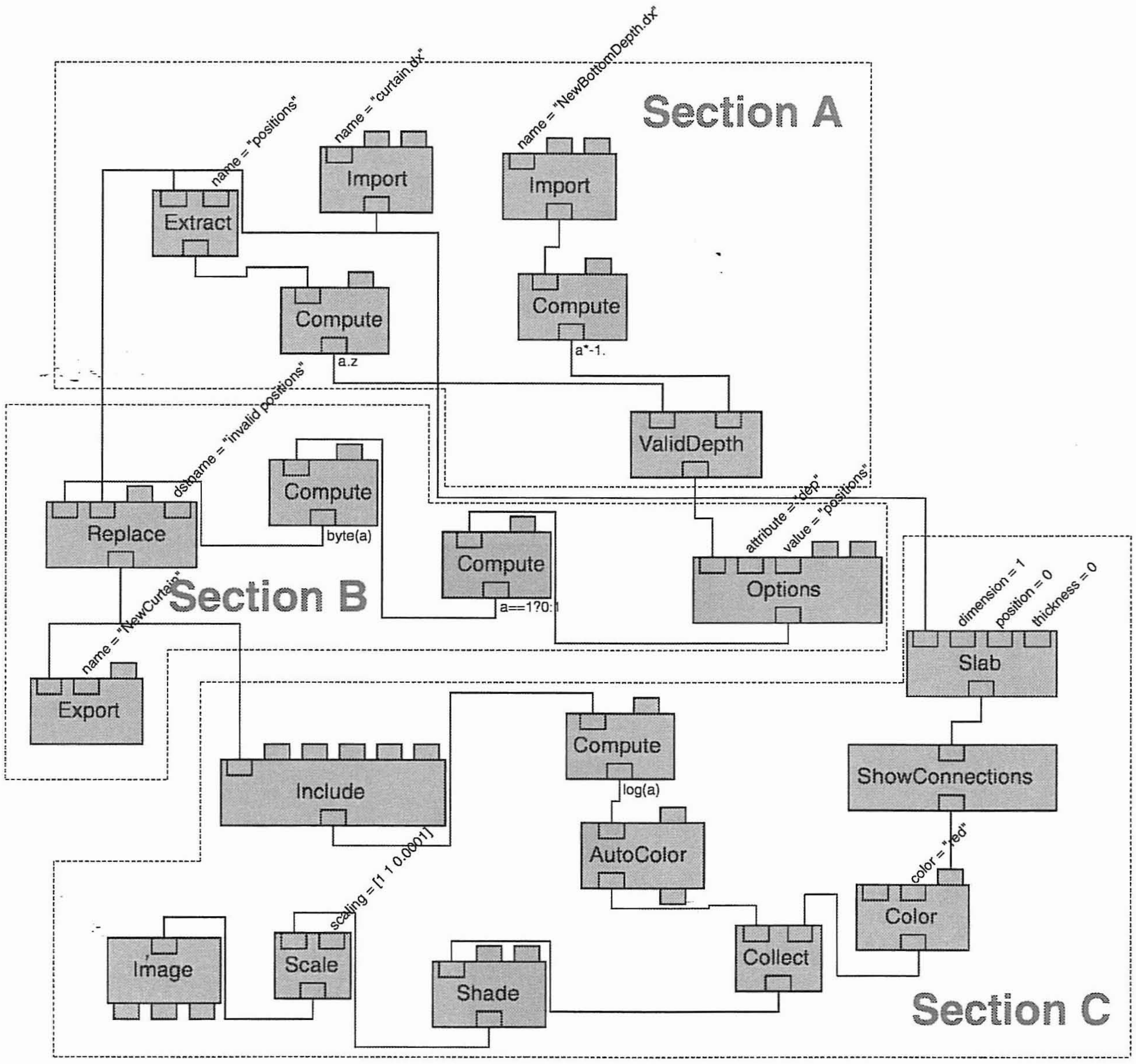
Appendix Fig. 5. The `MakeNewBottomDepth.net` program.



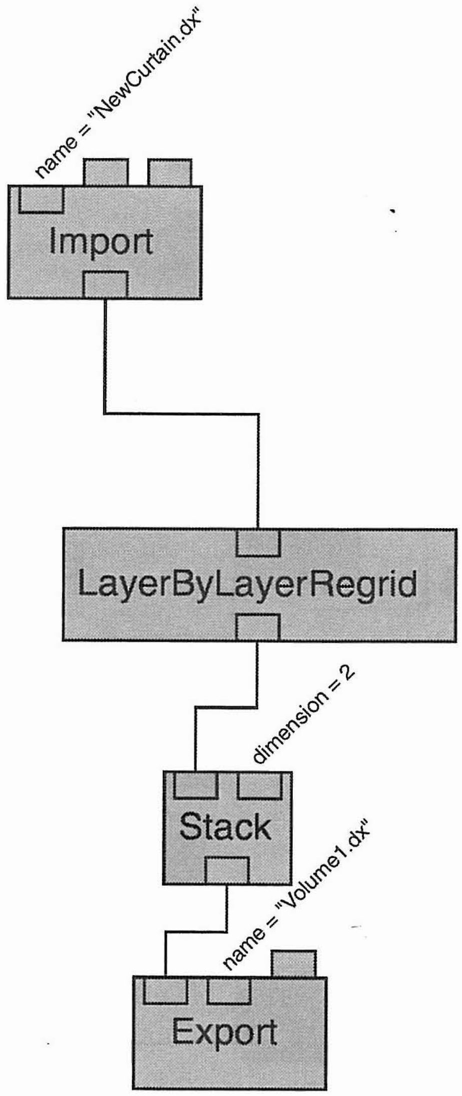
Appendix Fig. 6. The `ShipTrack3D.net` program.



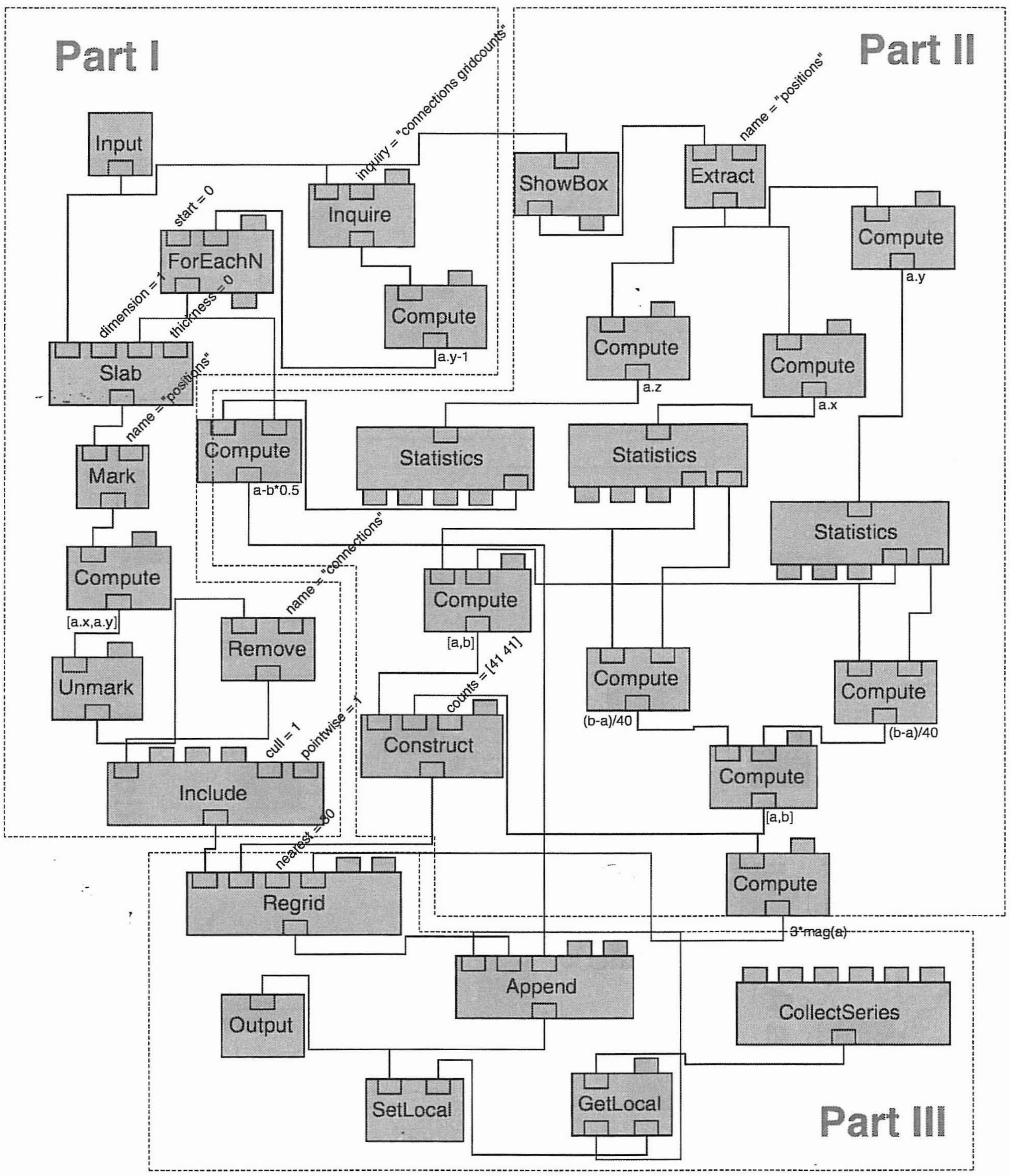
Appendix Fig. 7. The `MakeCurtain.net` program.



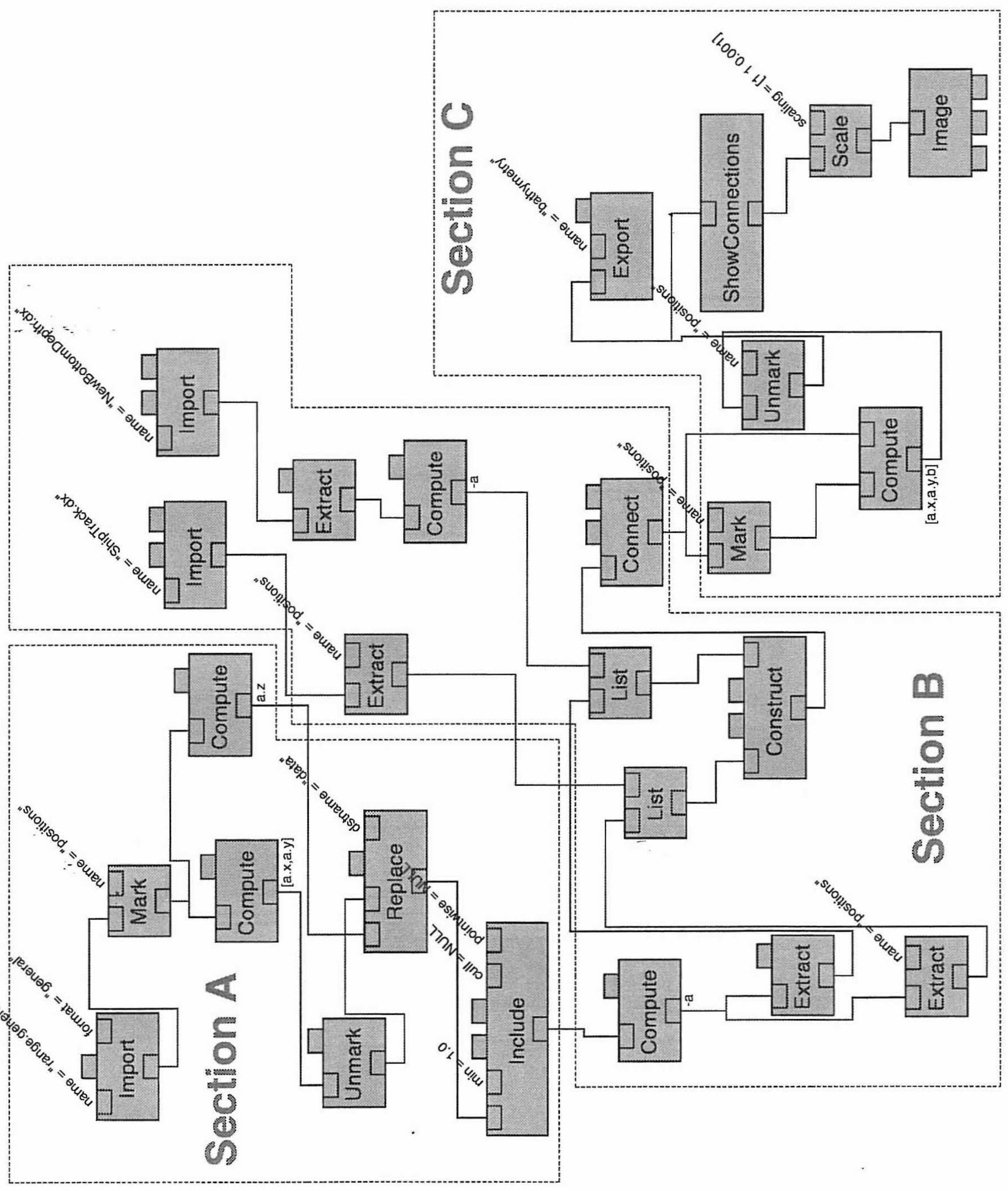
Appendix Fig. 8. The `MakeNewCurtain.net` program.



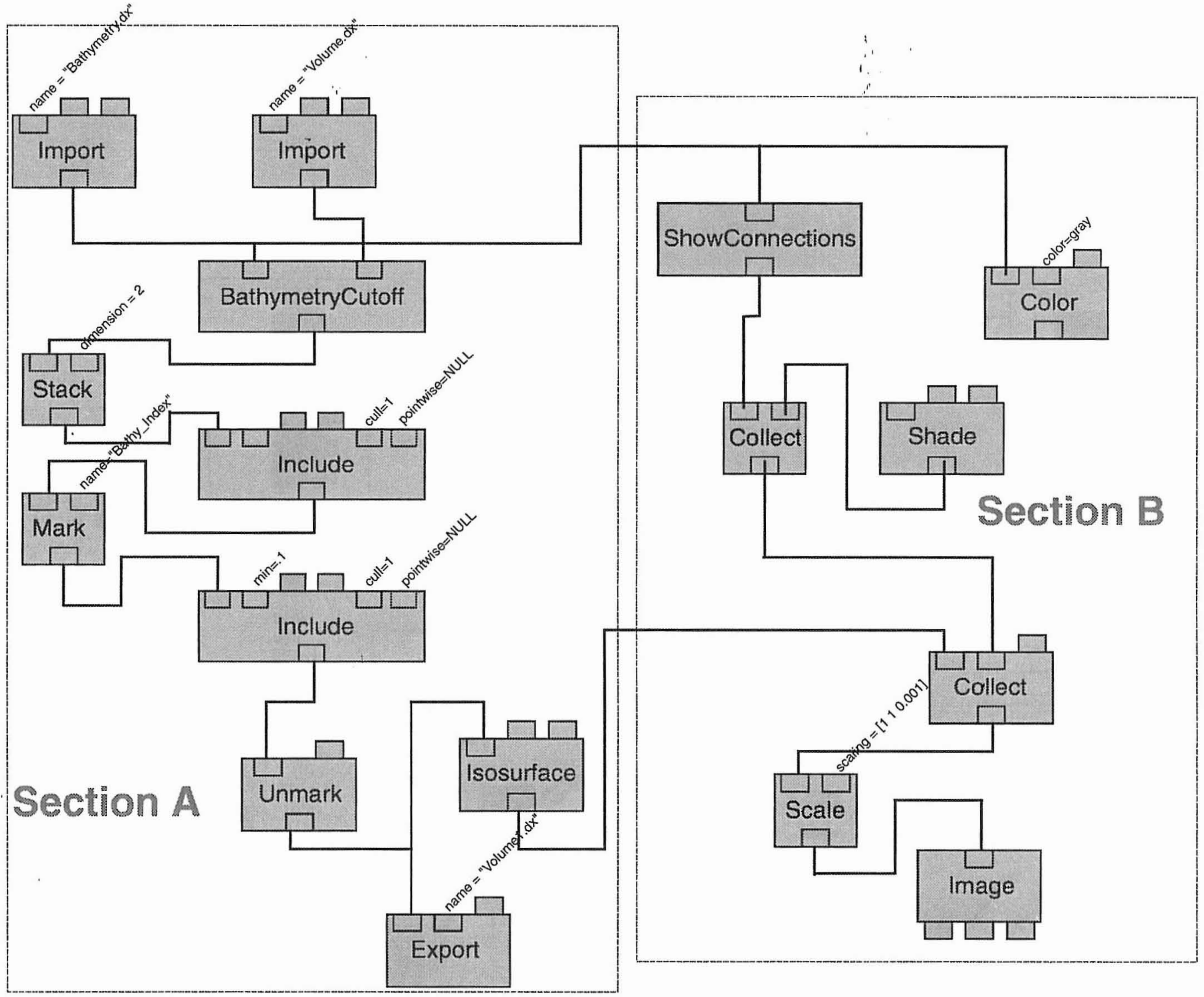
Appendix Fig. 9. The `MakeVolume.net` program



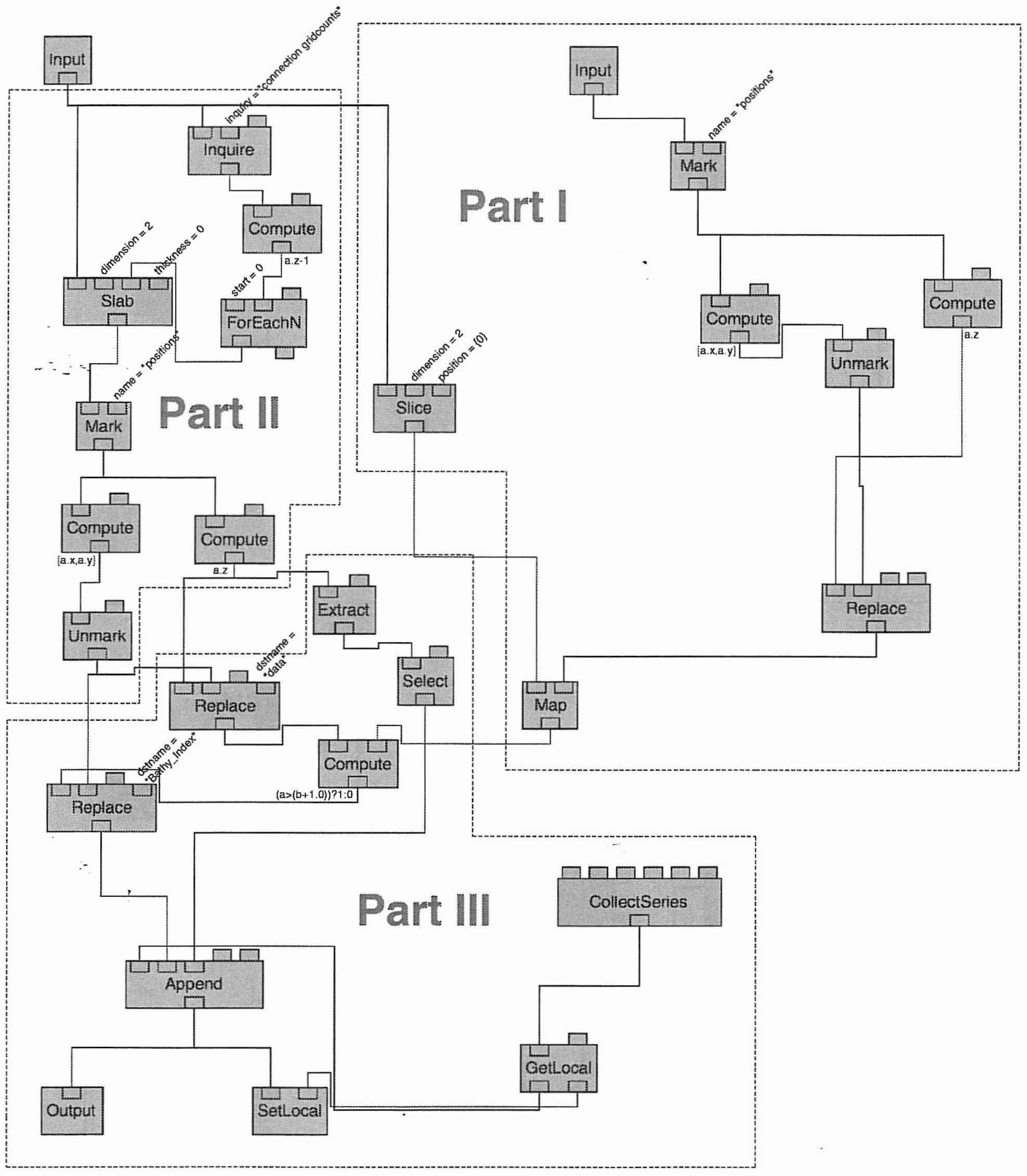
Appendix Fig. 10. The `LayerByLayerRegrid.net` macro.



Appendix Fig. 11. The `MakeBathymetryNewGrid.net` program.



Appendix Fig. 12. The `CutVolume.net` program.



Appendix Fig. 13. The `BathymetryCutoff.net` macro.