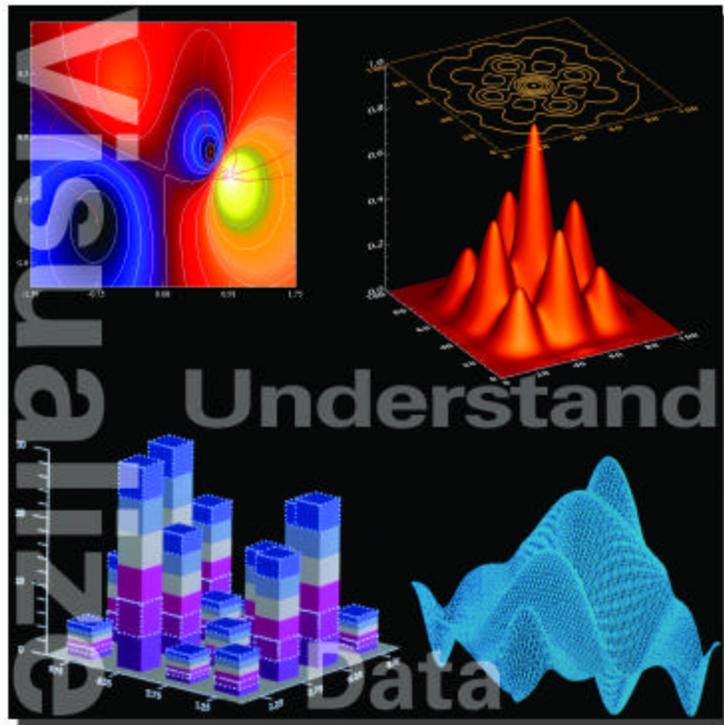




P V - W A V E 7 . 5[®]



G T - G R I D U s e r ' s G u i d e

HELPING CUSTOMERS **SOLVE** COMPLEX PROBLEMS

Visual Numerics, Inc.

Visual Numerics, Inc.
2500 Wilcrest Drive
Suite 200
Houston, Texas 77042-2579
United States of America
713-784-3131
800-222-4675
(FAX) 713-781-9260
<http://www.vni.com>
e-mail: info@boulder.vni.com

Visual Numerics, Inc.
7/F, #510, Sect. 5
Chung Hsiao E. Rd.
Taipei, Taiwan 110 ROC
+886-2-727-2255
(FAX) +886-2-727-6798
e-mail: info@vni.com.tw

Visual Numerics S.A. de C.V.
Cerrada de Berna 3, Tercer Piso
Col. Juarez
Mexico, D.F. C.P. 06600
Mexico

Visual Numerics, Inc. (France) S.A.R.L.
Tour Europe
33 place des Corolles
Cedex 07
92049 PARIS LA DEFENSE
FRANCE
+33-1-46-93-94-20
(FAX) +33-1-46-93-94-39
e-mail: info@vni-paris.fr

Visual Numerics International GmbH
Zettachring 10
D-70567 Stuttgart
GERMANY
+49-711-13287-0
(FAX) +49-711-13287-99
e-mail: info@visual-numerics.de

Visual Numerics, Inc., Korea
Rm. 801, Hanshin Bldg.
136-1, Mapo-dong, Mapo-gu
Seoul 121-050
Korea

Visual Numerics International, Ltd.
Suite 1
Centennial Court
East Hampstead Road
Bracknell, Berkshire
RG 12 1 YQ
UNITED KINGDOM
+01-344-458-700
(FAX) +01-344-458-748
e-mail: info@vniuk.co.uk

Visual Numerics Japan, Inc.
Gobancho Hikari Building, 4th Floor
14 Gobancho
Chiyoda-Ku, Tokyo, 102
JAPAN
+81-3-5211-7760
(FAX) +81-3-5211-7769
e-mail: vda-spirt@vnij.co.jp

© 1990-2001 by Visual Numerics, Inc. An unpublished work. All rights reserved. Printed in the USA. 2001

Information contained in this documentation is subject to change without notice.

IMSL, PV-WAVE, Visual Numerics and PV-WAVE Advantage are either trademarks or registered trademarks of Visual Numerics, Inc. in the United States and other countries.

The following are trademarks or registered trademarks of their respective owners: Microsoft, Windows, Windows 95, Windows NT, Fortran PowerStation, Excel, Microsoft Access, FoxPro, Visual C, Visual C++ — Microsoft Corporation; Motif — The Open Systems Foundation, Inc.; PostScript — Adobe Systems, Inc.; UNIX — X/Open Company, Limited; X Window System, X11 — Massachusetts Institute of Technology; RISC System/6000 and IBM — International Business Machines Corporation; Java, Sun — Sun Microsystems, Inc.; HPGL and PCL — Hewlett Packard Corporation; DEC, VAX, VMS, OpenVMS — Compaq Computer Corporation; Tektronix 4510 Rasterizer — Tektronix, Inc.; IRIX, TIFF — Silicon Graphics, Inc.; ORACLE — Oracle Corporation; SPARCstation — SPARC International, licensed exclusively to Sun Microsystems, Inc.; SYBASE — Sybase, Inc.; HyperHelp — Bristol Technology, Inc.; dBase — Borland International, Inc.; MIFF — E.I. du Pont de Nemours and Company; JPEG — Independent JPEG Group; PNG — Aladdin Enterprises; XWD — X Consortium. Other product names and companies mentioned herein may be the trademarks of their respective owners.

IMPORTANT NOTICE: Use of this document is subject to the terms and conditions of a Visual Numerics Software License Agreement, including, without limitation, the Limited Warranty and Limitation of Liability. If you do not accept the terms of the license agreement, you may not use this documentation and should promptly return the product for a full refund. Do not make illegal copies of this documentation. No part of this documentation may be stored in a retrieval system, reproduced or transmitted in any form or by any means without the express written consent of Visual Numerics, unless expressly permitted by applicable law.

Table of Contents

Preface [v](#)

- Intended Audience [v](#)
- Typographical Conventions [vi](#)
- Technical Support [vi](#)
 - FAX and E-mail Inquiries [vii](#)
 - Electronic Services [viii](#)

Chapter 1: Introduction [1](#)

- Starting PV-WAVE:GTGRID [2](#)
- Stopping PV-WAVE:GTGRID [2](#)
- New Features [2](#)
- Using PV-WAVE:GTGRID [3](#)
- PV-WAVE:GTGRID Approach to Gridding [3](#)
 - Primary Estimates of Z Values [4](#)
 - Secondary Estimates of Z Values [4](#)
 - Smoothing [6](#)
- The PV-WAVE:GTGRID Demonstration Program [6](#)
- System Limitations and Tips [6](#)
 - Digital UNIX Restrictions [6](#)
 - PV-WAVE:GTGRID 2.0 and the PV-WAVE Process Size [7](#)

Chapter 2: Reference [9](#)

- GTDESTROYSURF Procedure [9](#)
 - Usage [9](#)
 - Input Parameters [9](#)
 - Keywords [9](#)
 - Discussion [9](#)
- GTGETBOUNDS Function [10](#)

Usage 10
Input Parameters 10
Returned Value 10
Keywords 10
Discussion 11

GTGETFAULTS Function 11

Usage 11
Input Parameters 11
Returned Value 11
Keywords 11
Discussion 11

GTGETGRID Function 12

Usage 12
Input Parameters 12
Returned Value 12
Input Keywords 12
Output Keywords 12
Discussion 12

GTGETSURFS Function 13

Usage 13
Output Parameters 13
Returned Value 13
Keywords 13

GTGETVALUE Function 14

Usage 14
Input Parameters 14
Returned Value 14
Keywords 14
Discussion 14

GTGRID Function 15

Usage 15
Input Parameters 15
Keywords 15
Discussion 18
Method Keyword Arguments 19

GTINSTALLSURF Function 22

- Usage [22](#)
- Input Parameters [22](#)
- Returned Value [22](#)
- Keywords [23](#)
- Discussion [24](#)

GTRESTORESUF Function 25

- Usage [25](#)
- Input Parameters [25](#)
- Returned Value [25](#)
- Keywords [25](#)
- See Also [25](#)

GTSAVESURF Procedure 26

- Usage [26](#)
- Input Parameters [26](#)
- Keywords [26](#)
- Discussion [26](#)
- See Also [26](#)

***Appendix A: Examples* [A-1](#)**

- Example 1** [A-1](#)
- Example 2** [A-4](#)
- Example 3** [A-6](#)
- Example 4** [A-8](#)
- Example 5** [A-10](#)
- Example 6** [A-11](#)
- Example 7** [A-14](#)
- Example 8** [A-16](#)
- Example 9** [A-18](#)
- Example 10** [A-20](#)
- GT_GET_DATA File Listing** [A-22](#)
- GT_CON_DATA File Listing** [A-24](#)

Appendix B: Error Messages **B-1**

GTGRID Error Messages **B-1**

GTGETGRID Error Messages **B-7**

GTGETVALUE Error Messages **B-7**

GTINSTALLSURF Error Messages **B-8**

GTRESTORESUF and GTSAVESURF Error Messages **B-13**

GTGRID Index **1**

Preface

This guide explains how to use PV-WAVE:GTGRID 2.0; it contains the following parts:

- **Preface** — Describes the contents of this guide, the typographical conventions used, and how to obtain customer support.
- **Chapter 1, *Introduction*** — Introduces PV-WAVE:GTGRID and explains the PV-WAVE:GTGRID approach to gridding.
- **Chapter 2, *Reference*** — Detailed descriptions of the PV-WAVE:GTGRID functions.
- **Appendix A, *Examples*** — Presents several examples showing how to use PV-WAVE:GTGRID.
- **Appendix B, *Error Messages*** — Describes several PV-WAVE:GTGRID error messages and suggests how to correct the errors.
- ***GTGRID Index***

Intended Audience

This guide is for the experienced PV-WAVE user. Only minimal information on using PV-WAVE is covered here. Some knowledge of gridding procedures and techniques is helpful.

This guide does not assume familiarity with the GTGRID Library.

Typographical Conventions

You will find the following conventions used throughout this manual:

- Code examples appear in this typeface. For example:

```
PLOT, temp, s02, Title = 'Air Quality'
```

- Code comments are preceded by a semicolon and are shown in this typeface, immediately below the commands they describe. For example:

```
PLOT, temp, s02, Title = 'Air Quality'  
; This command plots air temperature data vs. sulphur  
; dioxide concentration.
```

- Variables are shown in lowercase italics (*myvar*), function and procedure names are shown in uppercase (XYOUTS), keywords are shown in mixed case italic (*XTitle*), and system variables are shown in regular mixed case type (!Version). For better readability, all GUI development routines are shown in mixed case (WwMainMenu).

Technical Support

If you have problems installing, unlocking, or running your software, contact Visual Numerics Technical Support by calling:

Office Location	Phone Number
Corporate Headquarters Houston, Texas	713-784-3131
Boulder, Colorado	303-939-8920
France	+33-1-46-93-94-20
Germany	+49-711-13287-0
Japan	+81-3-5211-7760
Korea	+82-2-3273-2633
Mexico	+52-5-514-9730
Taiwan	+886-2-727-2255
United Kingdom	+44-1-344-458-700

Users outside the U.S., France, Germany, Japan, Korea, Mexico, Taiwan, and the U.K. can contact their local agents.

Please be prepared to provide the following information when you call for consultation during Visual Numerics business hours:

- Your license number, a six-digit number that can be found on the packing slip accompanying this order. (If you are evaluating the software, just mention that you are from an evaluation site.)
- The name and version number of the product. For example, PV-WAVE 7.0.
- The type of system on which the software is being run. For example, SPARCstation, IBM RS/6000, HP 9000 Series 700.
- The operating system and version number. For example, HP-UX 10.2 or IRIX 6.5.
- A detailed description of the problem.

FAX and E-mail Inquiries

Contact Visual Numerics Technical Support staff by sending a FAX to:

Office Location	FAX Number
Corporate Headquarters	713-781-9260
Boulder, Colorado	303-245-5301
France	+33-1-46-93-94-39
Germany	+49-711-13287-99
Japan	+81-3-5211-7769
Korea	+82-2-3273-2634
Mexico	+52-5-514-4873
Taiwan	+886-2-727-6798
United Kingdom	+44-1-344-458-748

or by sending E-mail to:

Office Location	E-mail Address
Boulder, Colorado	support@boulder.vni.com
France	support@vni-paris.fr
Germany	support@visual-numerics.de
Japan	vda-sprt@vnij.co.jp
Korea	support@vni.co.kr
Taiwan	support@vni.com.tw
United Kingdom	support@vniuk.co.uk

Electronic Services

Service	Address
General e-mail	info@boulder.vni.com
Support e-mail	support@boulder.vni.com
World Wide Web	http://www.vni.com
Anonymous FTP	ftp.boulder.vni.com
FTP Using URL	ftp://ftp.boulder.vni.com/VNI/
PV-WAVE Mailing List:	Majordomo@boulder.vni.com
To subscribe include:	subscribe pv-wave YourEmailAddress
To post messages	pv-wave@boulder.vni.com

Introduction

PV-WAVE:GTGRID™ gives PV-WAVE users additional interpolation and extrapolation power by providing access to a library of gridding routines prepared by Geophysical Techniques (GT), Inc.

The GTGRID Library is a structured collection of FORTRAN routines for solving surface estimation (gridding) problems. It is especially valuable for PV-WAVE users in such areas as:

- Oil and Gas Industry
- Aerospace
- Radar and Sonic Applications
- Weather and Atmospheric Conditions

The core function of PV-WAVE:GTGRID, the GTGRID function, accepts 3D random data from PV-WAVE in the form of x , y , and z vectors. The result from the GTGRID function is a 2D array containing the gridded z values on a uniform x - y grid.

Other functions in the PV-WAVE:GTGRID suite allow you to manipulate and subset gridded data, and work with surface data.

The PV-WAVE:GTGRID functions are described in [Chapter 2, Reference](#).

Starting PV-WAVE:GTGRID

To start PV-WAVE:GTGRID do the following:

- At the `WAVE>` prompt, enter the following command to load and initialize PV-WAVE:GTGRID:

```
WAVE> @gtgrid_startup
```

- Once you see the following message, you are ready to use PV-WAVE:GTGRID:

```
PV-WAVE:GTGRID is initialized
```

Stopping PV-WAVE:GTGRID

If PV-WAVE:GTGRID is loaded and you want to exit PV-WAVE:GTGRID, perform the following procedure:

- At the `WAVE>` prompt, enter the following command to unload PV-WAVE:GTGRID.

```
WAVE> @gtgrid_unload
```

Unloading returns your system to the state it was in before using PV-WAVE:GTGRID by doing the following three things:

- It unloads PV-WAVE:GTGRID functions from memory.
- It returns the PV-WAVE:GTGRID license to the license manager, freeing the license up for others to use.
- It deletes all common variables in `GTGRID_COMMON`.

New Features

GTGRID 2.0 offers the following improvements:

- Improved gridding results.
- Increased control over gridding parameters.
- New functions for saving, restoring, accessing, and sampling grid information.
- Ability to sample gridded results in the context of the original gridded vectors, *x* and *y*.

- Increased the number of points allowed for the *Direct* method of gridding from 10,000 to 100,000.

NOTE Creases are not supported in PV-WAVE:GTGRID 2.x. GTGRID function calls that use creases will run without error; however, the creases are ignored.

Using PV-WAVE:GTGRID

The general steps to using PV-WAVE:GTGRID are:

- Decide how to get your data into PV-WAVE. For more information on input/output methods and routines, refer to the PV-WAVE documentation.
- Choose the gridding method that best suits your data. You specify the method to use with the GTGRID function *Method* keyword. The methods include *Direct*, *Scatter*, *Cluster*, and *Weighted*. Refer to [Chapter 2, Reference](#) for detailed information on the keywords and methods associated with the GTGRID function.
- Execute the GTGRID function and other PV-WAVE:GTGRID functions from the PV-WAVE command line or in a PV-WAVE application you have written.

NOTE When PV-WAVE:GTGRID functions are running, messages are displayed related to the processing. These messages are informational only, and do not require any action or affect the operation of PV-WAVE:GTGRID in any way.

PV-WAVE:GTGRID Approach to Gridding

Gridding refers to methods that interpolate a set of random points onto a grid mesh. This grid mesh can then be used to represent a surface that establishes the spatial relationships between the random points.

The GTGRID function provides you with several unique gridding methods, each designed to treat a certain class of data in an optimum manner. The choice of gridding method is made primarily based on the distribution of the input points.

The representation of gridded data within PV-WAVE:GTGRID is actually a matrix of z values, where each row and column index corresponds to some value of y and x respectively. Given a set of observations, usually randomly placed in x - y space,

the GTGRID function generates estimates of the z values at points, called nodes, on a grid.

You control the values of x and y by specifying the desired ranges of x and y and the number of grid nodes along each axis. Once a grid definition is specified, the problem of generating the z values at each node can involve up to three stages, depending on the gridding method you choose. These stages include:

- Primary estimates
- Secondary estimates
- Smoothing

These stages are discussed in the following sections.

Primary Estimates of Z Values

The first stage employed by the GTGRID function in producing a gridded dataset is to calculate primary estimates of the z values. The primary estimates attempt to determine z values and map the data points onto specified grid points. GTGRID calculates primary gridding estimates in the immediate vicinity of the input points, using the input points as control.

Because all the GTGRID methods are designed to create a single-valued function of two variables, some averaging of the input points will occur whenever the independent variables are very close to each other. Using the *Method* keyword you can specify how the primary estimates are calculated — by the *Direct*, *Scatter*, *Cluster*, or *Weighted* method.

Generally, you select a method that is based on the distribution of the random points for computing a grid or surface. The presence of faults will also determine which methods you can use.

Secondary Estimates of Z Values

The second stage employed by the GTGRID function in producing a gridded dataset is to calculate secondary estimates of z values. Note that secondary estimates are only calculated when the *Scatter*, *Cluster*, and *Weighted* methods are used. Secondary estimates are not calculated when the *Direct* method is used (that is, when the keyword `Method = 'Direct'`).

Secondary estimates are calculated at grid nodes that were too remote to be considered in the primary estimates. The secondary estimates use the previously defined grid nodes as control. This secondary gridding process may be performed several times, each time using the previously computed values as additional input.

When secondary estimates are calculated, significant improvements in the resulting surface can be obtained. The *Scatter*, *Cluster*, and *Weighted* methods all perform secondary estimates. The *Direct* method does not.

GTGRID improves the secondary estimates by effectively doubling the grid interval when computing grid nodes that are not adjacent to input points. This reduces the computations by a factor of eight to one. In other words, you can use four times as many grid nodes as you would on another system for about the same amount of computer time.

To compute secondary estimates, a procedure very similar to the *Scatter* method is used. The main differences between secondary estimates and the *Scatter* method are:

- Grid nodes are used instead of input points and,
- The local surface is not constrained to honor any particular point.

NOTE Only those grid nodes adjacent to a defined grid node are evaluated when secondary estimates are calculated. Thus for every calculation there is a point very close in proximity. This process is performed over the entire grid, the new values added to the grid, and then the process is repeated until all grid nodes have been evaluated or found to be impossible to compute.

Another significant item to note regarding secondary estimates is that only every other node in each direction is computed. This avoids a great deal of redundant calculation. Nodes that were skipped are then quickly filled using a bi-cubic interpolation scheme.

Processing Fault Data

As a simple definition, a fault can be considered a gross discontinuity of data. A simple example might be the break in continuity in the slope of a roof where it joins a wall.

GTGRID accepts fault values in the form of x -fault and y -fault vectors, which are specified as arguments to the *Xfault* and *Yfault* keywords.

To define faults to be used by GTGRID, each fault point is assumed to be connected to its predecessor unless it is a terminator. A terminator is defined as an x - y pair of 0.0, 0.0. The next point following a terminator is a starting point for a new sequence of fault points.

Connected fault points define the location of lines where discontinuities are desired in the interpolation. If a sequence of points is closed, the interior of the polygon

described is considered ‘null’ or undefined. The total number of fault segments cannot exceed 15,000. Faults must always be expressed in the same coordinate system as the random data.

Processing Creases

Creases are not supported in PV-WAVE:GTGRID version 2.0.

Smoothing

Smoothing seeks to minimize curvature of the surface by adjusting the secondary estimates. You control the amount of smoothing (number of smoothing passes) with the *Nsmooth* keyword. The greater the number of smoothing passes, the closer the resulting surface is to the solution of the biharmonic equation, which approximates a thin metal plate deformed into the desired shape.

Each smoothing pass modifies those secondary grid estimates that do not conform to the minimum curvature model based on its adjacent grid node neighbors. (The minimum curvature model refers the deformation of a “thin metal plate” so that it passes through data points. The deformation produced is defined by how much force is required to “push” the plate to an elevation of z at a particular point.) These updated values are then used as input values to the next pass. This recursive, numerical solution to the biharmonic equation converges slowly, but usually will give acceptable results with as few as 10 passes.

The PV-WAVE:GTGRID Demonstration Program

You can view a demonstration of GTGRID by running the program `wgtdemo.pro`. You can look at this program’s source code to obtain ideas for writing your own GTGRID applications.

This program is located in the directory:

```
$VNI_DIR/gtgrid-2_0/lib
```

System Limitations and Tips

Digital UNIX Restrictions

The current PV-WAVE:GTGRID 2.0 Digital UNIX release is slightly restricted. Problems will occur under Digital UNIX if you try to access memory beyond the

lower $2^{32} - 1$ locations. GTGRID contains a trap for this event and execution stops if this occurs. Visual Numerics will release a full 64-bit Digital UNIX version of PV-WAVE:GTGRID 2.0 in the near future. Please contact your customer support representative for current information.

PV-WAVE:GTGRID 2.0 and the PV-WAVE Process Size

If you are working with large data sets and producing large grids, we recommend that you destroy surfaces after creating them. To do this, set the *Nodestroy* keyword to zero (0) in the GTGRID function call. (`Nodestroy=0` is the default.) Destroying surfaces helps to control the growth of the PV-WAVE process size.

The amount of memory allocated for gridding is strongly related to the dimensions of the requested grid (see the *Nx* and *Ny* keywords of the GTGRID function). Very large grid dimensions (200-by-200 or greater) require PV-WAVE to allocate a substantial amount of memory. Once allocated to the PV-WAVE process, that memory cannot be returned to the operating system until you exit PV-WAVE.

Destroying GTGRID surface information allows the allocated memory to be reused by the current PV-WAVE process for future GTGRID calls or for other variables.

If you are producing grids with large dimensions using the GTGRID function, we recommend that you maintain constant grid dimensions in successive GTGRID function calls. If the grid dimensions are kept constant, and the surfaces are destroyed, GTGRID can more effectively reuse memory that is already allocated to the PV-WAVE process. This is true because the grid is stored in a contiguous block of memory. Changing the grid dimensions in successive GTGRID calls can cause memory fragmentation and consume much more memory than if the dimensions were kept constant in successive calls.

Finally, in working with large data sets, it is helpful to have a substantial amount of memory and swap space available on your machine.

Reference

This chapter describes the PV-WAVE:GTGRID functions and procedures. After installing and initializing PV-WAVE:GTGRID, you can access the GTGRID functions from the PV-WAVE command line or use them in a PV-WAVE application.

GTDESTROYSURF Procedure

Releases all allocated memory space associated with a specified surface.

Usage

GTDESTROYSURF [, *surf_id*]

Input Parameters

surf_id — (optional) (long) A valid ID for a gridded surface.

Keywords

All — If nonzero, destroys all active surfaces. (Default: 0)

Discussion

The value of *surf_id* is obtained from the *Surf_id* keyword in the GTGRID function or from the output of the GTINSTALLSURF function.

GTGRID provides access to a library of gridding routines prepared by Geophysical Techniques (GT), Inc. The GT libraries maintain an internal representation of generated grids and their attributes such as the faults, the bounding polygon, the spacing between nodes, the origin, and so on. The default behavior for the GTGRID function is to destroy (free the memory for) this internal GT surface representation after gridding is complete. Destroying this surface representation does not affect any PV-WAVE variables, but after a surface is destroyed, it can no longer be queried by functions such as GTGETVALUE and GTGETGRID.

If you unload the GTGRID option (GTGRID_UNLOAD), all of the active surfaces are destroyed.

GTGETBOUNDS Function

Retrieves the 3D bounding box from a gridded surface.

Usage

result = GTGETBOUNDS(*surf_id*)

Input Parameters

surf_id — (long) A valid ID for a gridded surface.

Returned Value

result — A 2D floating-point array containing the bounding box of the grid surface.

Keywords

Xmax — (float) Returns the maximum *x* value of the bounding box.

Xmin — (float) Returns the minimum *x* value of the bounding box.

Ymax — (float) Returns the maximum *y* value of the bounding box.

Ymin — (float) Returns the minimum *y* value of the bounding box.

Zmax — (float) Returns the maximum *z* value of the bounding box.

Zmin — (float) Returns the minimum *z* value of the bounding box.

Discussion

The value of *surf_id* is obtained from the *Surf_id* keyword in the GTGRID function or from the output of the GTINSTALLSURF function.

If you have lost the information on the *x*, *y*, and *z* vectors that were used to generate a given grid, the surface ID for that grid can be used with GTGETBOUNDS to obtain the original maxima and minima of your grid vectors. GTGETBOUNDS is also useful after restoring a surface from a file (GTRESTORESUF) to obtain the minima and maxima of the grid vectors for the surface.

GTGETFAULTS Function

Retrieves the fault lines from a surface.

Usage

result = GTGETFAULTS(*surf_id*)

Input Parameters

surf_id — (long) A valid ID for a gridded surface.

Returned Value

result — A 2D floating-point array containing the fault lines associated with the gridded surface.

Keywords

Maxpts — (long) Limits the number of points in the faults array.

Num_faults — (long) Returns the number of faults in the input variable.

Discussion

The value of *surf_id* is obtained from the *Surf_id* keyword in the GTGRID function or from the output of the GTINSTALLSURF function.

Use GTGETFAULTS to regenerate fault vectors that were used to generate the grid for that surface. For example, after restoring a surface from a file (GTRESTORESUF) use GTGETFAULTS to obtain the fault vectors that were used to generate the grid.

GTGETGRID Function

Retrieves the grid matrix from a surface.

Usage

```
result = GTGETGRID(surf_id)
```

Input Parameters

surf_id — (long) A valid ID for a gridded surface.

Returned Value

result — A 2D floating-point array containing the grid matrix associated with the gridded surface.

Input Keywords

Nx — (integer) Specifies the *x*-dimension size of the returned grid. If *Nx* is not specified, but *Ny* is specified, then *Nx* is set equal to *Ny*. (Default: 20)

Ny — (integer) Specifies the *y*-dimension size of the returned grid. If *Ny* is not specified, but *Nx* is specified, then *Ny* is set equal to *Nx*. (Default: 20)

Output Keywords

Grid_nx — (integer) Returns the actual number of rows in the original grid.

Grid_ny — (integer) Returns the actual number of columns in the original grid.

Discussion

The value of *surf_id* is obtained from the *Surf_id* keyword in the GTGRID function or from the output of the GTINSTALLSURF function.

Use GTGETGRID to obtain the 2D grid after restoring a surface from a file (GTRESTORESUF). GTGETGRID can also be used to obtain different grid dimensions than were originally used to generate the surface description. For example, if the original grid was generated using $nx=ny=20$, a 50-by-50 grid can be obtained using GTGETGRID and the surface ID from the 20-by-20 grid.

NOTE Using GTGETGRID with $nx=ny=50$ is not the same as defining $nx=ny=50$ during a GTGRID call. When extracting a larger dimensioned grid from a surface with a smaller dimensioned grid, the larger grid will be centered in the grid array and points beyond the original grid dimensions will be set to the null value for that grid.

GTGETSURFS Function

Lists the IDs of all currently active surfaces.

Usage

result = GTGETSURFS([*num_surfs*])

Output Parameters

num_surfs — (optional) (long) The number of active surfaces.

Returned Value

result — A 1D long array containing the IDs of the currently active surfaces.

Keywords

None.

GTGETVALUE Function

Returns the value at a specified coordinate on the surface.

Usage

result = GTGETVALUE(*surf_id*, *xvalue*, *yvalue*)

Input Parameters

surf_id — (long) A valid ID for a gridded surface.

xvalue — (float) An *x*-coordinate value on the surface.

yvalue — (float) A *y*-coordinate value on the surface.

Returned Value

result — (float) The value of the surface at the specified point.

Keywords

Nulval — (float) Specifies a value to be substituted for an “undefined” point.

Discussion

The value of *surf_id* is obtained from the *Surf_id* keyword in the GTGRID function or from the output of the GTINSTALLSURF function.

GTGETVALUE accepts either scalar or array values for the *x* and *y* coordinate points. You can use GTGETVALUE to draw a line along the top of a surface drawing. First, create a surface using GTGRID or GTINSTALLSURF. Then generate an array of *z* values for *x*, *y* points along the surface using GTGETVALUE. To do this, you can use the original *x*, *y* vectors from your GTGRID call, or use FINDGEN to generate new *x* and *y* vectors. Then, use the *z* values returned by GTGETVALUE to overplot a line on the surface plot of your grid.

GTGRID Function

Produces an evenly sampled grid from scattered data.

Usage

result = GTGRID(*xvec*, *yvec*, *zvec*)

Input Parameters

- *xvec* — An array of floating-point values containing the *x* coordinates of the random input data.
- *yvec* — An array of floating-point values containing the *y* coordinates of the random input data.
- *zvec* — An array of floating-point values containing the *z* coordinates of the random input data.

Keywords

Bounding_Poly — A 2D floating-point array specifying the vertices of a polygon describing the outer boundary of the resulting grid. The first and last points of the polygon must be identical.

Dist_Avg — (float) Specifies the distance over which to average input points before using them to compute the grid nodes. In general, the value of *Dist_Avg* is a fraction of the grid spacing (*Xspacing/Yspacing* keywords), but you can minimize the number of points processed by increasing the value of *Dist_Avg*.
(Default: $\text{MIN}([xspacing, yspacing]) / 2$)

Interp_Dist — (float) Specifies the maximum distance to interpolate from each point. (Default: $\text{MAX}([xmax-xmin, ymax-ymin]) * \text{radius} / 200.0$) Where *radius* is expressed as a percentage. See the *Radius* keyword.

Method — The *Method* keyword is a string value used to specify how primary gridding estimates at grid nodes are performed. Primary estimates are discussed in the section *Primary Estimates of Z Values* on [page 4](#). The arguments for the *Method* keyword are: *Direct*, *Scatter*, *Cluster*, and *Weighted*. These arguments are discussed in the section *Method Keyword Arguments* on [page 19](#).

Neighbors — (integer) Specifies the number of octants required to have neighbors before calculation of a node is attempted. For extrapolation, use `Neighbors=3`. (Default: 5)

Nodestroy — If nonzero, preserves the surface information. By default, the memory allocated for the surface information is freed.

Nomessage — If nonzero, messages that are output while the GTGRID function is running are not displayed. By default, messages are displayed.

Nsmooth — (integer) Specifies the number of smoothing passes made on the data. The default is 20. For more information on smoothing, see [Smoothing on page 6](#).

Nulval — (float) Specifies the numeric value used by the gridded for null or missing grid nodes in the resulting output grid. The default value is determined by inspecting the `zvec` array and computing:

$$\text{Nulval} = \min(\text{ZVEC}) - (\max(\text{ZVEC}) - \min(\text{ZVEC}))$$

This is designed to ensure a reasonable presentation of the output grid by PV-WAVE commands such as SURFACE.

Nx — (integer or long) Specifies the number of columns in the result array. If `Nx` is not specified, but `Ny` is specified, then `Nx` is set equal to `Ny`. (Default: 20)

Ny — (integer or long) Specifies the number of rows in the result array. If `Ny` is not specified, but `Nx` is specified, then `Ny` is set equal to `Nx`. (Default: 20)

Radius — (float) Specifies the distance, expressed as a percentage of the maximum of `Nx` and `Ny`, over which interpolation will be performed. This keyword is used by all the gridding methods except for *Direct*. (Default: 50 percent)

One interpretation of this parameter is to consider it as the distance at which the covariance function goes to zero. Points that are farther away from each other than this are essentially independent.

A larger value of *Radius* is occasionally required when the input points are distributed such that large gaps exist in the coverage. A larger value of *Radius* will slow processing down.

A smaller value of *Radius* is useful for speeding things up, and can be used to generate surfaces that have holes wherever there is no coverage. As a general rule, grid nodes that are farther away from an input point than *Radius* will not be assigned.

Surf_id — (long) Returns a value used to reference surface information internal to the GTGRID libraries. This value is used as input to other PV-WAVE:GTGRID functions and procedures. Use the *Nodestroy* keyword with the *Surf_id* keyword to preserve surface information in memory for future access.

Xfaults — A 1D array of floating-point values used to define faults relative to the *xvec* parameter. The values for *Xfaults* must be defined within the same coordinate system as the random data. There is no default for this keyword. If *Xfaults* is present, but *Yfaults* is not provided, an error message is produced and GTGRID returns the scalar 0.0 as the results.

Figure 2-1 represents an interpolated curve with a fault.

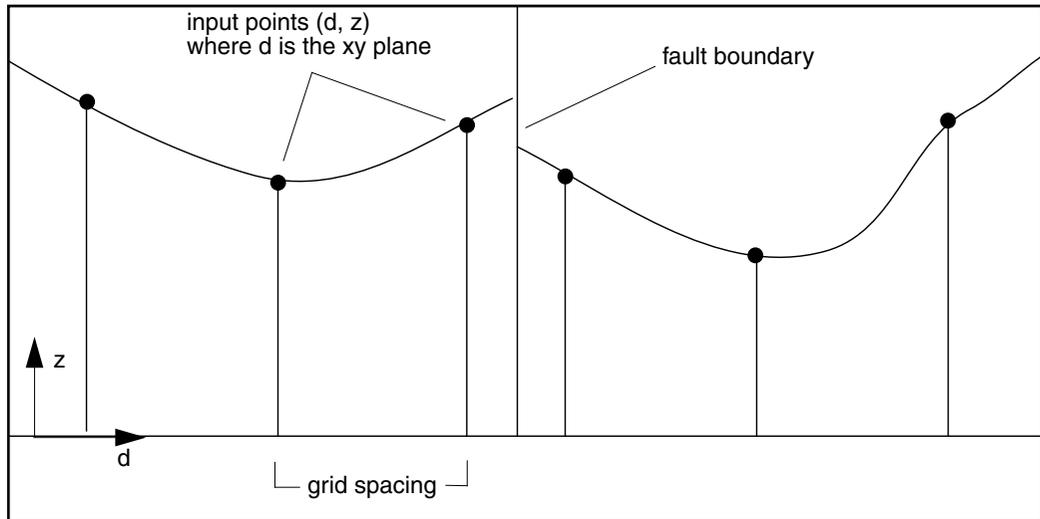


Figure 2-1 Interpolated curve, shown in two dimensions, produced using the Cluster method with a fault.

Xmax — (float) Defines the maximum *x* clipping plane in data coordinates. The default is the maximum value found in *xvec*.

Xmin — (float or double) Defines the minimum *x* clipping plane in data coordinates. The default is the minimum value found in *xvec*.

Xorg — (float) Specifies the *x*-coordinate of the origin of the generated grid. (Default: *xmin* (if defined); otherwise MIN(*xvec*))

Xspacing — (float) Specifies the spacing between consecutive nodes in the *x* direction. (Default: $(x_{max} - x_{min}) / (n_x - 1)$)

Yfaults — A 1D array of floating-point values used to define faults relative to the *yvec* parameter. The values of *Yfaults* must be defined within the same coordinate system as the random data. There is no default for this keyword. If *Yfaults* is present, but *Xfaults* is not provided, an error message is issued and GTGRID returns the scalar 0.0 as the result.

Ymax — (float) Defines the maximum *y* clipping plane in data coordinates. The default value is the maximum value found in *yvec*.

Ymin — (float or double) Defines the minimum *y* clipping plane in data coordinates. The default is the minimum value found in *yvec*.

Yorg — (float) Specifies the *y*-coordinate of the origin of the generated grid. (Default: `MIN(yvec)`)

Yspacing — (float) Specifies the spacing between consecutive nodes in the *y* direction. (Default: $(y_{max} - y_{min}) / (ny - 1)$)

Zresolution — (float) Specifies the minimum meaningful difference in *z* values during gridding. (Default: the value of the *Nulval* keyword)

Discussion

The range covered by the grid is determined from the values of the *Nx*, *Ny*, *Xorg*, *Yorg*, *Xspacing*, and *Yspacing* keywords as follows:

From (*Xorg*) to ($Xorg + (Nx - 1) * Xspacing$)

From (*Yorg*) to ($Yorg + (Ny - 1) * Yspacing$)

If you generate many large grids, without destroying the internal GT surface storage, you may run out of memory. Therefore, it is wise to let GTGRID destroy your surfaces if you are calling it successively while adjusting your gridding parameters.

Once you are satisfied with the gridding parameters, use the */Nodestroy* keyword; the surface ID of that generated surface can then be used by other PV-WAVE:GTGRID 2.0 functions.

When you are finished working with a surface, you can destroy it using the GTDESTROYSURF procedure. If you do not know the ID of your surface, you can use the GTGETSURFS function to obtain a list of the active GT surfaces.

TIP Because GTGRID must compute the *Xmax*, *Xmin*, *Ymax* and *Ymin* values, if you do not provide them, you can improve overall performance time by supplying values for these keywords.

TIP GTGRID allows you to define the clipping planes on the input data using the keywords *Xmin*, *Xmax*, *Ymin* and *Ymax*. Once the gridding process is complete and the PV-WAVE prompt is displayed, defining the clipping planes in the *z* range can be accomplished with PV-WAVE commands. If the *z* values on return are known

to be in the range of 1000 to 9000, but the values of interest are above 5000, then the following command could be used:

```
WAVE> Gtgrid_results = Gtgrid_results > $ 5000
```

In essence, this command sets a lower z clipping plane of 5000. Similarly, both the upper and lower clipping planes can be specified in a similar command:

```
WAVE> Gtgrid_results = 3000 > $  
      Gtgrid_results < 7000
```

Method Keyword Arguments

The following table summarizes the advantages and disadvantages of each method and the types of data best suited to each method. The methods are described in detail in following sections.

For an example of a GTGRID call that uses the *Method* keyword, see examples 5 and 7 in [Appendix A, Examples](#).

Method	Advantages	Disadvantages	Intended Data
Direct	Fast. Visually pleasing results.	Only works for a small data sets (< 100000 points). Performs excessive averaging on larger data sets	Small data sets with no discontinuities (faults).
Scatter	Handles faults and large data sets well. Honors local gradient.	Not appropriate for noisy data — If data is noisy, spurious results can occur.	Random, large data sets with or without discontinuities.
Cluster	Good for use with data that is close together or noisy. Ignores excessive gradients implied by closely spaced noisy data.	Excessive averaging of closely spaced points is appropriate only for noisy data.	Randomly distributed noisy data with or without discontinuities.
Weighted	Fast. Good for visualizing input data with a small radius. Gives a “quick look”.	No gradient calculations are made.	Very large data sets.

Method = 'Direct'

The Direct method provides an approximation for generating a minimum curvature solution. This method can only be used on data that contains less than 100,000 points. It can not be used with fault data. Use of this approach is recommended wherever it is possible. It is the fastest gridding method for small data sets, and also tends to produce the most visually pleasing results.

This method employs point averaging whenever the number of points exceeds 1000. This may cause the resulting surface to not pass directly through the input points that were averaged out.

All nodes are calculated regardless of the input point distribution. The resulting surface as represented by the z values at each node in the grid is the exact solution to a set of equations that approximates the relationships among a thin metal plate, a set of desired displacements at the random input points, and the forces applied at each point.

By solving this system of equations the necessary values of the force are determined at each point. These values are then used to compute the plate displacement at every grid node. If no averaging is performed, this solution provides a surface that passes through every point specified and has the least amount of curvature possible, as shown in [Figure 2-2](#).

NOTE If averaging was performed to reduce the number of points, the solution will not necessarily pass through those points that were averaged.

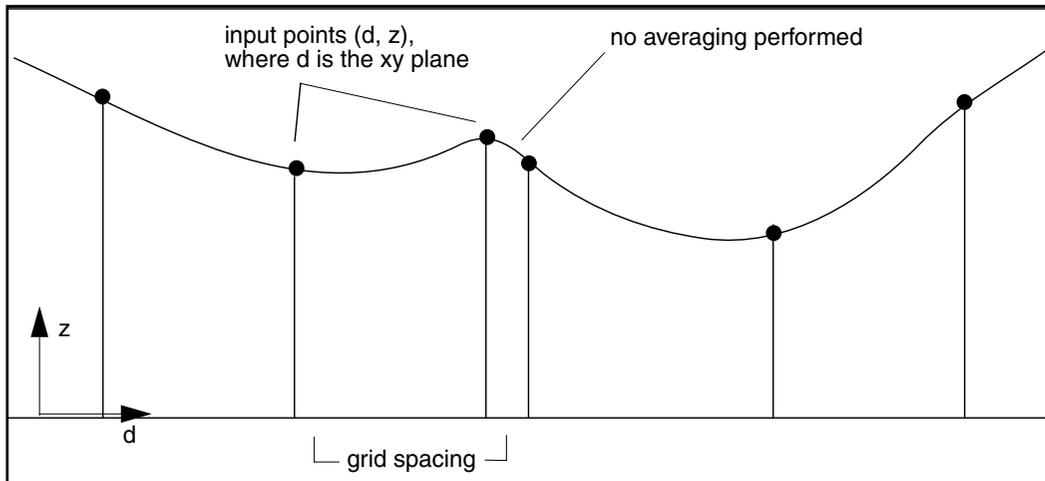


Figure 2-2 Interpolated curve, shown in two dimensions, using the Direct method. No averaging is performed.

Method = ‘Scatter’ (Default)

The Scatter method uses the Radial Search for Scattered Points algorithm. This is the default method for GTGRID.

The radial search uses randomly positioned input points to compute grid nodes in the “immediate vicinity” of each input point.

The radial search technique relies on localized fits of a plane to the selected subsets of the input points. Using each input point as an origin, a subset of the surrounding points is formed by selecting the nearest two points in each octant about the origin. Each point is then weighted according to its displacement from the origin. The plane is also constrained to pass through the point selected as the origin.

If a successful fit is accomplished, the grid nodes about the point selected as an origin are then assigned z values.

Only grid nodes that are interior to the convex hull of the input data point distribution will be assigned a z value. (The convex hull is a boundary that encompasses all the data points. For example, if each data point were a nail in a board, the convex hull could be represented by a rubber band stretched around all the nails.) This is done to minimize edge effects that are inherent to this approach.

This process repeats using every input point as an origin. GTGRID attempts to assign all unassigned grid nodes during secondary gridding. Note that some grid nodes are computed more than once if the input point distribution is such that more than one point is contained within a grid cell.

Method = ‘Cluster’

The Cluster method uses the Radial Search for Clustered or Linear Data algorithm. The Cluster Method differs from the Scatter Method in that additional effort is employed during calculation of grid nodes near input points to minimize the extrapolation of excessive gradients due to noise or points that are very close together.

Therefore, it is recommended that this method be used especially if noisy or clustered data is being processed.

Method = ‘Weighted’

This method uses the Weighted algorithm. This is a relatively fast method of mapping points onto a specified grid without any attempt at interpolation or gradient estimation.

Use this method if you want to input a set of values which were computed on a grid by a modeling program or if an abundance of data is available relative to the grid

size desired. Each grid node is a weighted combination of the data points surrounding it within adjacent cells. Secondary gridding is then employed to fill unassigned grid nodes. If the input distribution is fairly uniform, this method is much faster than the others. [Figure 2-3](#) shows the curve fit produced with this method.

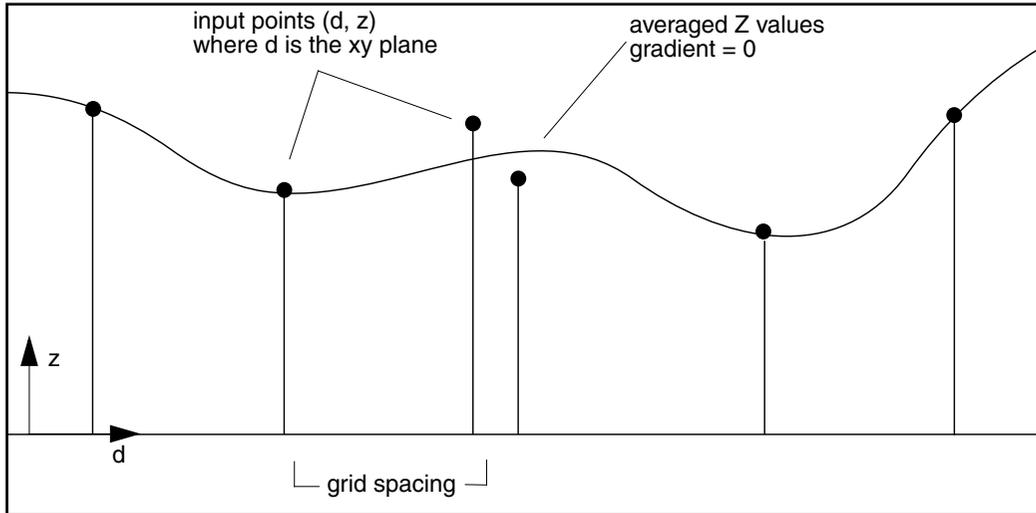


Figure 2-3 Interpolated curve, shown in two dimensions, produced using the Weighted method.

GTINSTALLSURF Function

Creates a surface from grid and fault lines.

Usage

result = GTINSTALLSURF(*thegrid*)

Input Parameters

thegrid — A 2D floating-point array of grid data.

Returned Value

result — (long) The surface ID for the installed surface.

Keywords

Bounding_Poly — A 2D floating-point array specifying a bounding polygon for the generated grid.

Nsmooth — (integer) Specifies the number of smoothing passes made on the data. (Default: 20)

Nulval — (float) Specifies the numeric value used by the gridder for null or missing grid nodes in the resulting output grid. The default value is determined by inspecting the *zvec* array and computing:

$$\text{Nulval} = \text{zmin} - (\text{zmax} - \text{zmin})$$

Numpts — (long) Specifies the number of points in the vectors from which the grid was formed. (Default: the maximum lengths of the keywords *Xvec*, *Yvec*, and *Zvec*)

Nx — (integer or long) Specifies the number of columns in the result array. If *Nx* is not specified, but *Ny* is specified, then *Nx* is set equal to *Ny*. (Default: 20)

Ny — (integer or long) Specifies the number of rows in the result array. If *Ny* is not specified, but *Nx* is specified, then *Ny* is set equal to *Nx*. (Default: 20)

Xfaults — A 1D array of floating-point values used to define faults relative to the *xvec* parameter. The values for *Xfaults* must be defined within the same coordinate system as the random data. There is no default for this keyword. If *Xfaults* is present, but *Yfaults* is not provided, an error message is produced and GTINSTALLSURF returns the scalar 0.0 as the results.

Xmax — (float) Defines the maximum *x* clipping plane in data coordinates. The default is the maximum value found in *xvec*.

Xmin — (float or double) Defines the minimum *x* clipping plane in data coordinates. The default is the minimum value found in *xvec*.

Xorg — (float) Specifies the *x*-coordinate of the origin of the generated grid. (Default: *Xorg* = *xmin*)

Xspacing — (float) Defines the spacing between consecutive nodes in the *x* direction. (Default: *Xspacing* = $(\text{xmax} - \text{xmin}) / (\text{nx} - 1)$)

Xvec — A 1D floating-point array containing the *x* vectors for the grid.

Yfaults — A 1D array of floating-point values used to define faults relative to the *yvec* parameter. The values of *Yfaults* must be defined within the same coordinate system as the random data. There is no default for this keyword. If *Yfaults* is present, but *Xfaults* is not provided, an error message is issued and GTINSTALLSURF returns the scalar 0.0 as the result.

Ymax — (float) Defines the maximum *y* clipping plane in data coordinates. The default value is the maximum value found in *yvec*.

Ymin — (float or double) Defines the minimum *y* clipping plane in data coordinates. The default is the minimum value found in *yvec*.

Yorg — (float) Specifies the *y*-coordinate of the origin of the generated grid. (Default: *Yorg* = *ymin*)

Yspacing — (float) Defines the spacing between consecutive nodes in the *y* direction. (Default: *Yspacing* = (*y_{max}* - *y_{min}*) / (*ny* - 1))

Yvec — A 1D floating-point array containing the *y* vectors for the grid.

Zmax — (float) Defines the maximum *z* clipping plane in data coordinates. The default is the maximum value found in *zvec*.

Zmin — (float or double) Defines the minimum *z* clipping plane in data coordinates. The default is the minimum value found in *zvec*.

Zresolution — (float) The minimum meaningful difference in *z* values during gridding.

Zvec — A 1D floating-point array containing the *z* vectors for the grid.

Discussion

GTINSTALLSURF can be used to install grids from GTGRID 1.x for use with the functions and procedures in GTGRID 2.0 which require a valid surface ID as input. GTINSTALLSURF must be able to determine the original grid *x* and *y* node spacing, the grid origin, and the null value for the grid. This information may be explicitly supplied to GTINSTALLSURF using the *Xspacing*, *Yspacing*, *Xorg*, *Yorg*, and *Nulval* keywords; or GTINSTALLSURF will calculate these values given the *x* vector from which the grid was formed or the *Xmin* and *Xmax* values, the *y* vector from which the grid was formed or the *Ymin* and *Ymax* values, and the *z* vector from which the grid was formed or the *Zmin* and *Zmax* values.

TIP Because GTINSTALLSURF must compute the *Xmax*, *Xmin*, *Ymax*, and *Ymin* values if you do not provide them, you can improve overall performance time by supplying values for these keywords.

GTRESTORESUF Function

Restores a previously saved surface.

Usage

result = GTRESTORESUF(*filename*)

Input Parameters

filename — (string) The name of a grid save file.

Returned Value

result — (long) The surface ID of the restored surface.

Keywords

None.

See Also

[GTSAVESURF](#)

GTSAVESURF Procedure

Saves a surface to disk.

Usage

GTSAVESURF, *surf_id*, *filename*

Input Parameters

surf_id — (long) A valid ID for a gridded surface.

filename — (string) The name of the save file.

Keywords

None.

Discussion

The value of *surf_id* is obtained from the *Surf_id* keyword in the GTGRID function or from the output of the GTINSTALLSURF function.

See Also

[GTRESTORESUF](#)

Examples

The best way to understand any new process is to look at numerous examples. We have provided many examples for you so that you can expertly apply your data needs to them.

NOTE Some gridding parameters used in these examples have little effect on the displayed surface; however, the effect of these parameters can be seen in the actual grid z -values or their statistical representations.

Example 1

This procedure demonstrates the use of the *Xorg*, *Yorg*, *Xmin*, *Xmax*, *Ymin*, *Ymax*, *Xspacing*, and *Yspacing* GTGRID keywords.

The data files are located in the directory:

```
$VNI_DIR/gtgrid-2_0/data
```

```
PRO GT20_EXAMPLE_1
```

```
    if ( !version.platform NE 'i386nt') THEN GT_GET_DATA, X, $  
        Y, Z, Xf, Yf
```

```
;
```

```
; Generate a grid with an x,y origin at 4200, 1000. The spacing between
; grid nodes in the x and y directions will be 50 and 150, respectively.
; The generated grid will be clipped in the X direction and shifted.
;
```

```
Zz = GTGRID(X, Y, Z, $
           xorg=4200.0, $
           yorg= 1000.0, $
           xspacing=50.0, $
           yspacing=150.0, $
           /nomessages)
```

```
;
; Display the generated grid.
;
```

```
SURFACE, Zz
```

```
HAK, /mesg
```

```
;
; Generate a grid with an x,y origin at 3000, 3000. Limit the X and Y
; vectors to a maximum of 5000. This cuts one side of the grid off
; because the yorg is greater than the minimum of Y.
;
```

```
Zz = GTGRID(X, Y, Z, $
           xorg = 3000.0, $
           yorg = 3000.0, $
           xmax = 5000.0, $
           ymax = 5000.0, $
           /nomessages)
```

```
;
; Display the generated grid.
;
```

```
SURFACE, Zz
```

```
HAK, /mesg
```

```

;
; Generate a grid with an x,y origin at 1000, 500. Limit the X
; vectors to a maximum of 5000. The generated grid will be shifted into
; the far corner of the grid array. The surface will also be surrounded
; by nulvals because the given x and y min/max ranges are greater than
; the mins/maxes of X and Y.
;
Zz = GTGRID(X, Y, Z, $
           xorg = 1000.0, $
           yorg = 500.0, $
           xmin = 2000.0, $
           ymin = 1000.0, $
           xmax = 6000.0, $
           ymax = 5000.0, $
           /nomessages)
;
; Display the generated grid.
;

SURFACE, Zz

RETURN

END ;; GT20_EXAMPLE_1

```

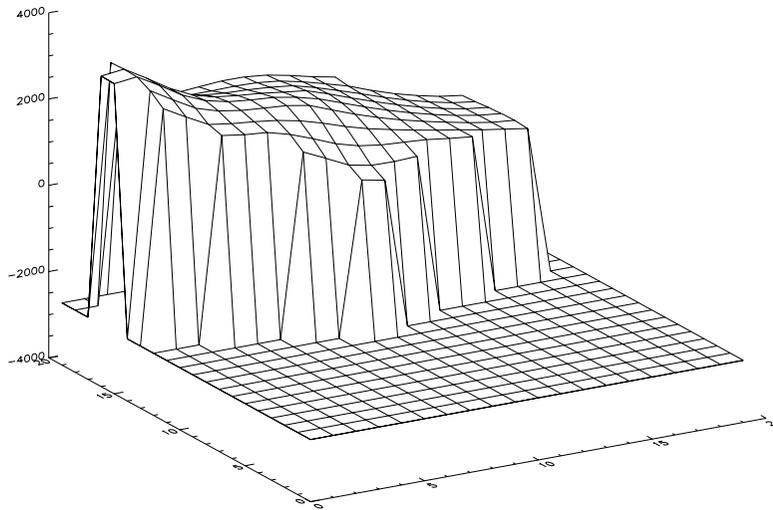


Figure A-1 The first grid generated in Example 1.

Example 2

This example procedure shows the effect of the GTGRID *Radius* keyword.

The data files are located in the directory:

```
$VNI_DIR/gtgrid-2_0/data
```

```
PRO GT20_EXAMPLE_2
```

```

if ( !version.platform NE 'i386nt' ) THEN GT_GET_DATA, X, $
Y, Z, Xf, Yf
;
; Define the radius of interpolation at 25% of the max of nx, ny grid space.
; Decreasing the radius to 25% does not produce a greatly noticeable effect
; in the generated grid. Looking at the values in Zz or the statistics
; min, max, avg, stddev of Zz will show that there is a slight difference in
; this grid and a grid generated with the default radius of 50%.
;
Zz = GTGRID(X, Y, Z, xf=Xf, yf=Yf, $
           radius = 25.0, $

```

```

                                /nomessages)

;
; Display the generated grid.
;
SURFACE, Zz

HAK, /Mesg

;
; Define the radius of interpolation at 1.5% of the max of nx,ny grid space.
; Since the default is 50%, the generated surface will be much more "rough"
; than for the default radius. There will be more undefined points in the
; grid due to lack of interpolation.
;

Zz = GTGRID(X, Y, Z, xf=Xf, yf=Yf, $
           radius = 1.5, $
           /nomessages)

;
; Display the generated grid.
;

SURFACE, Zz

RETURN

END ;; GT20_EXAMPLE_2

```

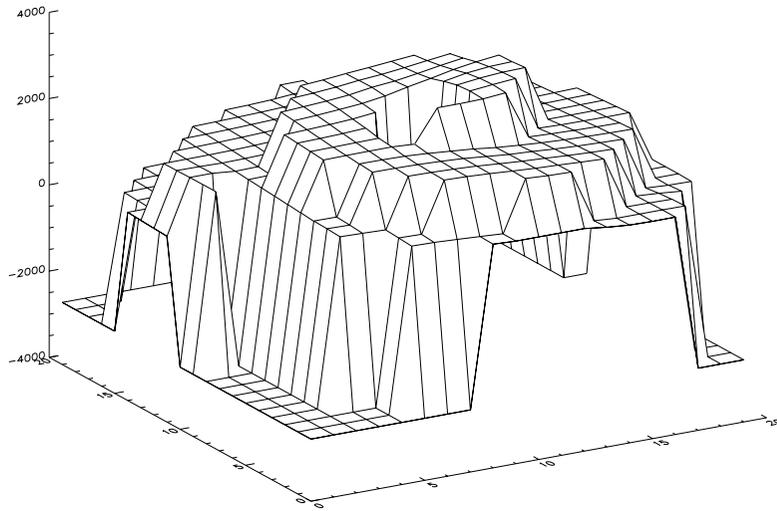


Figure A-2 The first grid generated in example 2.

Example 3

This procedure shows the effect of the GTGRID *Neighbor* keyword.

The data files are located in the directory:

```
$VNI_DIR/gtgrid-2_0/data
```

```
PRO GT20_EXAMPLE_3
```

```

if ( !version.platform NE 'i386nt') THEN GT_GET_DATA, X, $
Y, Z, Xf, Yf
;
; Define the spacing between nodes in the x and y directions at 100 and
; 125, respectively. Set the number of octants required to have neighbors
; before calculation of a node is attempted to 3 neighbors. This causes
; GTGRID to perform extrapolation, so some grid nodes that might be
; undefined (nulval) for a neighbors of 5 (the default), will now be
; assigned values.
;
Zz = GTGRID(X, Y, Z, $

```

```

        xspacing = 100.0, yspacing = 125.0, $
        neighbors = 3, $
        /nomessages)

;
; Display the generated grid.
;

SURFACE, Zz

HAK, /Mesg

;
; Define the spacing between nodes in the x and y directions at 100 and
; 125, respectively. Set the number of octants required to have neighbors
; before calculation of a node is attempted to 6 neighbors. This should
; make the generated grid a little more rough than for the default number
; of neighbors, 5.
;

Zz = GTGRID(X, Y, Z, $
        xspacing = 100.0, yspacing = 125.0, $
        neighbors = 6, $
        /nomessages)

;
; Display the generated grid.
;

SURFACE, Zz

RETURN

END ;; GT20_EXAMPLE_3

```

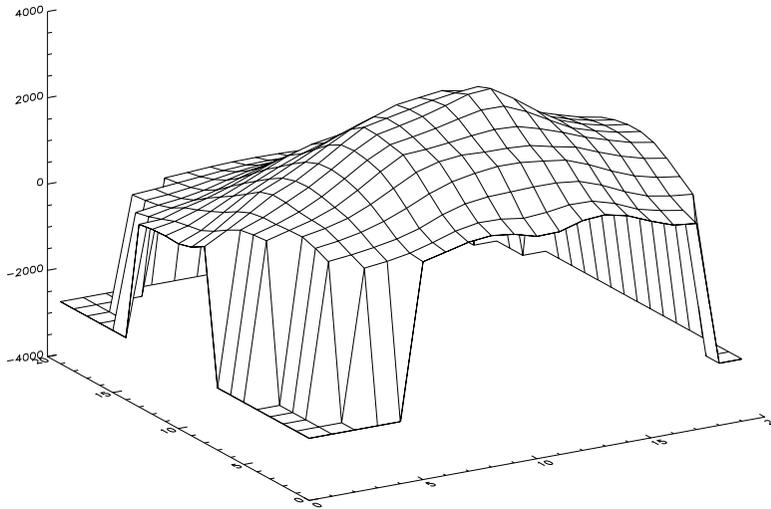


Figure A-3 The first grid generated in example 3.

Example 4

This procedure shows the effect of the *Dist_avg* and *Zresolution* GTGRID keywords.

The data files are located in the directory:

```
$VNI_DIR/gtgrid-2_0/data
```

```
PRO GT20_EXAMPLE_4
```

```
if ( !version.platform NE 'i386nt' ) THEN GT_GET_DATA, X, $
Y, Z, Xf, Yf
```

```
;
```

```
; Define the minimum meaningful difference in z values during gridding and
; averaging steps to be 50.0. The effects of this parameter is not apparent
; from looking at the grid. View the statistics min, max, stddev, avg of Zz
; to see the effect of zresolution. When the zresolution is equal to the nulval,
; it is ignored.
```

```
; Define the distance over which to average input points before using them
```

```
; to compute the grid nodes (dist_avg) to be 10. For this data, this will cause  
; input points to be averaged over a shorter distance than the default value  
; of MIN([xspacing,yspacing])/2.0, which is approximately 50.0 for this  
; particular data set.
```

```
;  
Zz = GTGRID(X, Y, Z, xf=Xf, yf=Yf, $  
           dist_avg = 10.0, $  
           zresolution = 50.0, $  
           /nomessages)
```

```
;  
; View the resulting grid.  
;
```

```
SURFACE, Zz
```

```
RETURN
```

```
END ;; GT20_EXAMPLE_4
```

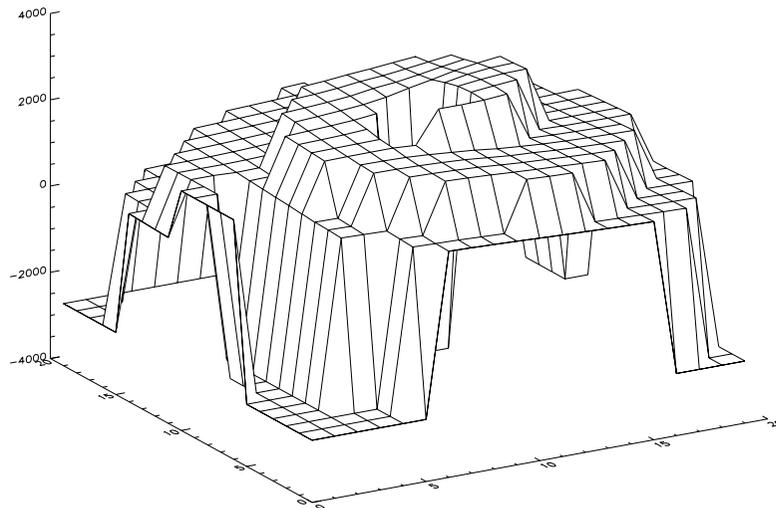


Figure A-4 Example 4 grid.

Example 5

This example procedure shows the effect of the GTGRID *Interp_dist* keyword. The procedure uses the *Cluster* method and returns the surface ID.

The data files are located in the directory:

```
$VNI_DIR/gtgrid-2_0/data
```

```
PRO GT20_EXAMPLE_5
```

```
if ( !version.platform NE 'i386nt') THEN GT_GET_DATA, X, $
Y, Z, Xf, Yf
;
; Set the maximum distance to interpolate from each point to (xmax-xmin)/20.0.
; This is a shorter distance than the default value which is based on the
; radius given. Therefore, there will be more undefined (nulval) points in
; the generated grid.
;
xmax = MAX(X, min=xmin)
Zz = GTGRID(X, Y, Z, xf=Xf, yf=Yf, $
           interp_dist = (xmax-xmin)/20.0, $
           surf_id = mysurf, $
           method = "CLUSTER", $
           /nomessages, /nodestroy)
;
; View the generated grid.
;
SURFACE, Zz
;
; Determine the value of the grid at the point 4045, 3032. To view the grid
; value for a series of x,y coordinates, pass 1D arrays to GTGETVALUE as the
; x and y parameters and a 1D array of grid values will be returned in ZzValue.
;
ZzValue = GTGETVALUE(mysurf, 4045.0, 3032.0)
```

```

GTDESTROYSURF, mysurf

PRINT, 'Value at 4045.0, 3032.0 = ', ZzValue

RETURN

END ;; GT20_EXAMPLE_5

```

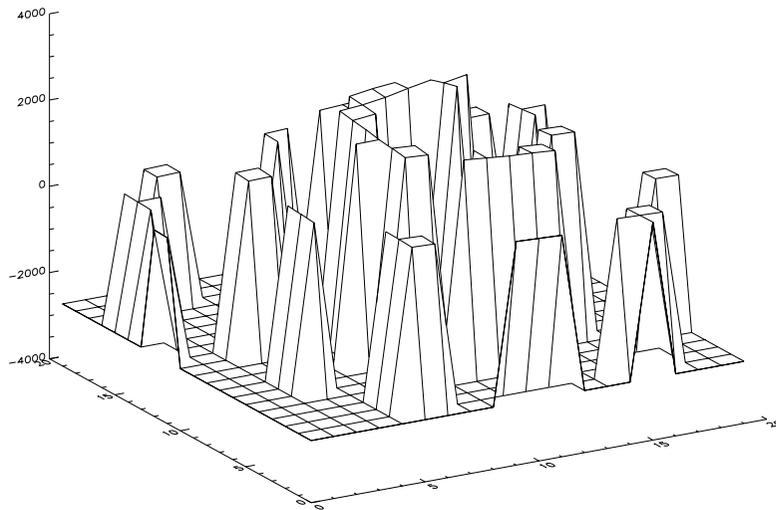


Figure A-5 Example 5 grid.

Example 6

This example procedure demonstrates the GTGETSURFS, GTGETGRID, and GTDESTROYSURF functions.

The data files are located in the directory:

```
$VNI_DIR/gtgrid-2_0/data
```

```
PRO GT20_EXAMPLE_6
```

```

if ( !version.platform NE 'i386nt') THEN GT_CON_DATA, X, $
Y, Z, Xf, Yf
;
; Generate two 50x75 grids. Do not destroy the surfaces after creation.
; One grid has a larger radius than the other, so it should be smoother and
; contain fewer undefined nodes.
;
Zz = GTGRID(X, Y, Z, xf=Xf, yf=Yf, $
           nx=50, ny=75, $
           neighbors=3, $
           radius = 25.0, $
           /nomessages, /nodestroy)

Zz = GTGRID(X, Y, Z, xf=Xf, yf=Yf, $
           nx=50, ny=75, $
           neighbors=3, $
           radius = 5.0, $
           /nomessages, /nodestroy)

;
; Get and array of valid surface IDs. And the number of active surfaces.
;
mysurfs = GTGETSURFS(num_surfs)

;
; Now display the grids that we generated by extracting them from the
; saved surfaces using the surface IDs.
;
FOR I=0, num_surfs-1 DO BEGIN
    Zz = GTGETGRID(mysurfs(I), nx=50, ny=75)
    SURFACE, Zz
    HAK, /Mesg
ENDFOR

;
; Let's clean up some memory by destroying those surfaces that we are

```

```

; no longer interested in.
;

FOR I=0, num_surfs-1 DO BEGIN
    GTDESTROYSURF, mysurfs(I)
ENDFOR

;
; Unless you had surfaces in memory from a previous GTGRID or GTINSTALLSURF
; or GTRESTORESURF call, there should not be any active surfaces now. Nosurfs
; will be -1 and num_nosurfs will be 0.
;
nosurfs = GTGETSURFS(num_nosurfs)

RETURN

END ;; GT20_EXAMPLE_6

```

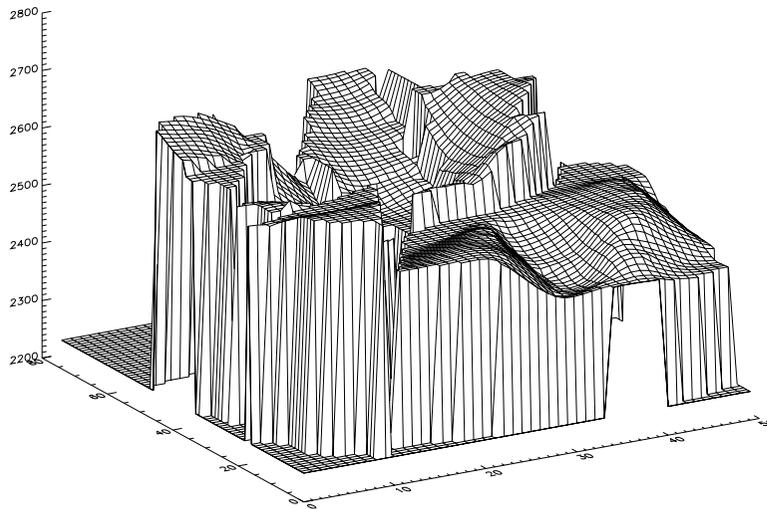


Figure A-6 Example 6 grid.

Example 7

This procedure demonstrates the GTGRID *Bounding_poly* keyword and the GTGETGRID and GTDESTROYSURF functions.

The data files are located in the directory:

```
$VNI_DIR/gtgrid-2_0/data
```

```
PRO GT20_EXAMPLE_7
```

```
if ( !version.platform NE 'i386nt') THEN GT_GET_DATA, X, $
Y, Z, Xf, Yf

;
; Define a bounding polygon to be the limits of x and y
;
xmin = MIN(X, max=xmax)
ymin = MIN(Y, max=yymax)
mypoly = fltarr(2,5)
mypoly(0,0)=xmin
mypoly(1,0)=ymin
mypoly(0,1)=xmin
mypoly(1,1)=ymax
mypoly(0,2)=xmax
mypoly(1,2)=ymax
mypoly(0,3)=xmax
mypoly(1,3)=ymin
mypoly(0,4)=xmin
mypoly(1,4)=ymin
;
; Generate a grid, Zz, using the cluster method. Define the distance over
; which to average input points before using them to compute grid nodes
; at 25.0. Don't destroy the surface and return the surface ID in the
; variable, mysurf.
;
Zz = GTGRID(X, Y, Z, xf=Xf, yf=Yf, $
```

```

        dist_avg=25.0, $
        bounding_poly=mypoly, $
        method='cluster', $
        surf_id=mysurf, /ndestroy, /nomessages)
;
; Display the generated grid.
;
SURFACE, Zz

HAK, /Mesg

;
; Now save the surface to a file. Filename is OS dependent. i.e. for an
; SGI running IRIX, the filename is gtsurfIRIX.srf.
;
filename = 'gtsurf'+!version.os+'.srf'

GTSAVESURF, mysurf, filename

;
; Extract the generated grid into a 50x50 array. Since the original grid
; was 20x20 (default nx and ny), points beyond the original dimensions will
; be assigned the nulval for the surface.
;
mynx = 50
myny = 50
mygrid = GTGETGRID(mysurf, nx=mynx, ny=myny)

;
; Display the extracted grid.
;
SURFACE, mygrid

;
; Let GTGRID free the memory for the surface.
;
GTDESTROYSURF, mysurf

```

```

RETURN

END ;; GT20_EXAMPLE_7

```

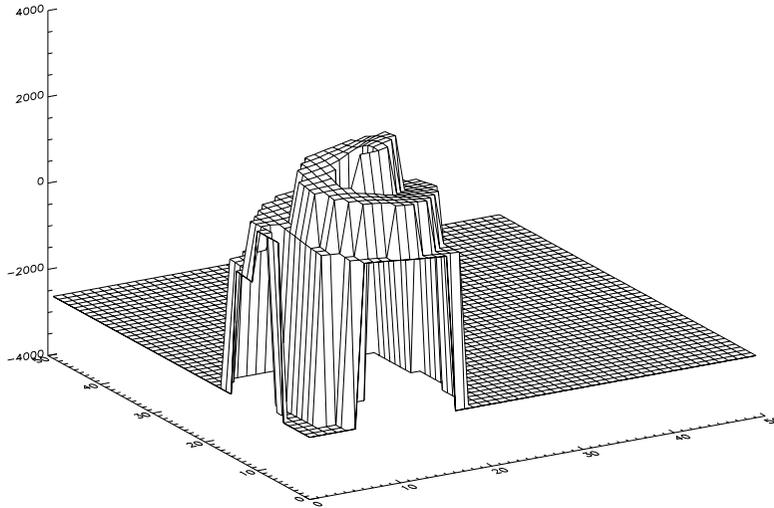


Figure A-7 Second grid in example 7.

Example 8

This procedure file that shows use of GTRESTORESUF, GTGETGRID, GTGETBOUNDS, GTGETFAULTS, and GTGETVALUE functions.

The data files are located in the directory:

```
$VNI_DIR/gtgrid-2_0/data
```

```

PRO GT20_EXAMPLE_8

;
; Open the surface file that was saved by Example 7.
;
filename = 'gtsurf'+!version.os+'.srf'

```

```

mysurf = GTRESTORESUF(filename)

;
; Extract the grid from the surface.
;
mygrid = GTGETGRID(mysurf)
;
; Extract the bounding box from the surface.
;
mybounds = GTGETBOUNDS(mysurf)
;
; Let GTGETBOUNDS return just the ymin value instead of the entire bounding
; box array.
;
ymin = GTGETBOUNDS(mysurf, /ymin)
;
; Extract the fault arrays (if there were any) from the surface.
;
myfaults = GTGETFAULTS(mysurf)
;
; Sample the grid at point 4022, 3055
;
myvalue = GTGETVALUE(mysurf, 4022.0, 3055.0)
;
; Let GTGRID free the memory allocated for this surface representation.

GTDESTROYSURF, mysurf

;
; Display the grid that was associated with the surface.
;
SURFACE, mygrid

RETURN

END ;; GT20_EXAMPLE_8

```

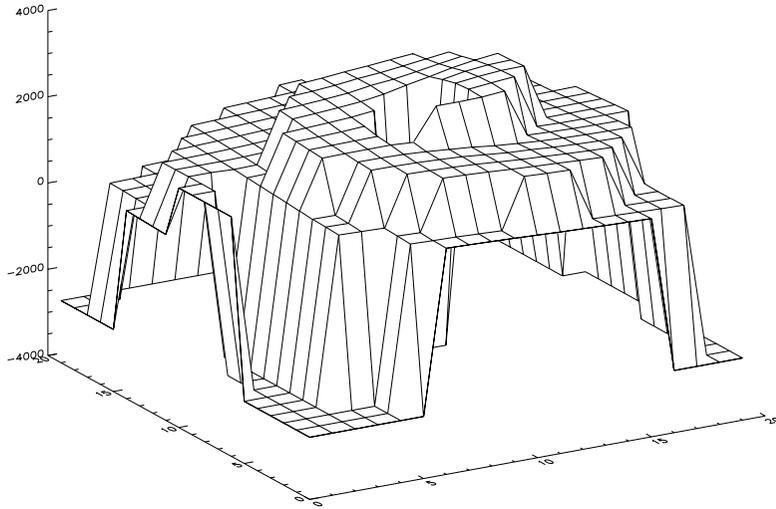


Figure A-8 Example 8 grid.

Example 9

This example procedure uses a larger grid size and the GTINSTALLSURF function.

The data files are located in the directory:

```
$VNI_DIR/gtgrid-2_0/data
```

```
PRO GT20_EXAMPLE_9
```

```
if ( !version.platform NE 'i386nt') THEN GT_CON_DATA, X, $
Y, Z, Xf, Yf
```

```
;
```

```
; Generate a grid that is 200x200 with sampling in the X and Y directions
```

```
; at 50.0 between nodes. Tell GTGRID not to destroy the surface.
```

```
;
```

```
Zz = GTGRID(X, Y, Z, xf=Xf, yf=Yf, $
           xspacing=50.0, yspacing=50.0, $
```

```

nx=200, ny=200, /nomessages, /nodelstroy)

;
; Display the generated grid.
;
SURFACE, Zz
;
; Find the active surface IDs.
;
mysurfs = GTGETSURFS(num_surfs)
print, 'Active surfs: ', mysurfs
;
; Install the grid we just generated as another surface.
;
surf_id = GTINSTALLSURF(Zz, xvec=X, yvec=Y, zvec=Z, xf=Xf,$
yf=Yf)
;
; Sample the grid using its new surface ID.
;
myvalue = GTGETVALUE(surf_id, 20478174.000, 4377234.500)
print, 'value at 20478174.000, 4377234.500 = ', myvalue
;
; Destroy all active surfaces.
;
GTDESTROYSURF,/all

RETURN

END ;; GT20_EXAMPLE_9

```

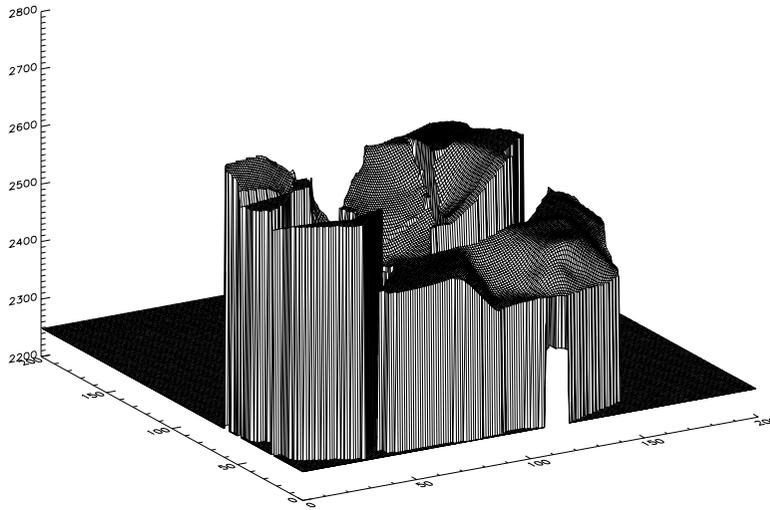


Figure A-9 Example 9 grid.

Example 10

This example procedure shows the use of the GTGRID *Min*, *Max*, *Nx*, and *Ny* keywords.

The data files are located in the directory:

```
$VNI_DIR/gtgrid-2_0/data
```

```
PRO GT20_EXAMPLE_10
```

```
if ( !version.platform NE 'i386nt' ) THEN GT_GET_DATA, X, $
Y, Z, Xf, Yf
```

```
;
```

```
; Define faults for the grid. xmin, xmax and ymin, ymax exceed the actual
; mins and maxes of X and Y so the grid will be surrounded on all sides
; by nulvals. Zz will be a 50x25 array.
```

```
;
```

```
Zz = GTGRID(X, Y, Z, xf=Xf, yf=Yf, $
xmin=2200.0, xmax=7000.0, $
```

```

        ymin= 1000.0, ymax=5000.0, $
        nx = 50, ny = 25, $
        /nomessages)
;
; Display the generated grid.
;

SURFACE, Zz

HAK, /Mesg

;
; Generate the same basic grid, but sample more finely in the Y space this
; time. Zz will be a 50x75 array.
;
Zz = GTGRID(X, Y, Z, xf=Xf, yf=Yf, $
        xmin=2200.0, xmax=7000.0, $
        ymin= 1000.0, ymax=5000.0, $
        nx = 50, ny = 75, $
        /nomessages)
;
; Display the generated grid.
;

SURFACE, Zz

RETURN

END ;; GT20_EXAMPLE_10

```

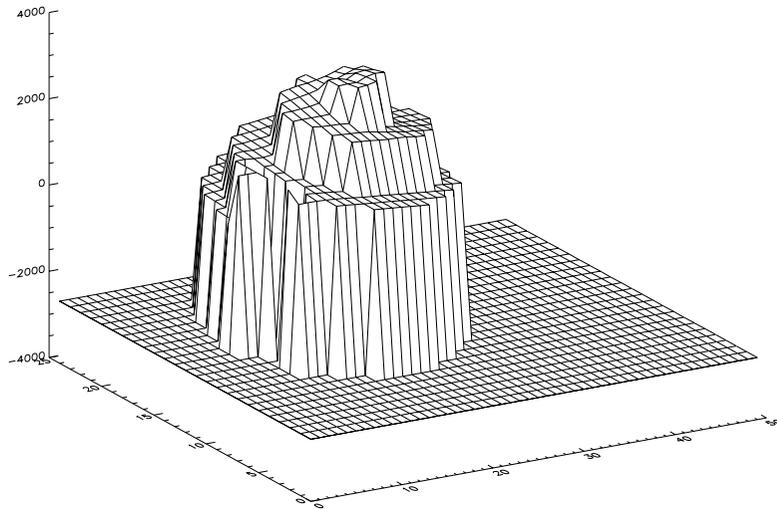


Figure A-10 Example 10 grid.

GT_GET_DATA File Listing

On a UNIX system, this file is located in the directory:

```
$VNI_DIR/gtgrid-2_0/lib
```

```
PRO Gt_get_data, X, Y, Z, Xf, Yf, Xc, Yc
```

```
;
```

```
; GT_GET_DATA: PV-WAVE procedure file to read the original data  
; files to be used for GTGRID testing. The file 'gt_fault.dat'  
; contains the data for faults, while 'gt_xyz.dat' contains the  
; X, Y and Z data.
```

```
;
```

```
; Invoke this procedure with the command:
```

```
; Gt_get_data, X, Y, Z, Xf, Yf, Xc, Yc
```

```
; where:
```

```
; X - returned as the X data from gt_xyz.dat  
; Y - returned as the Y data from gt_xyz.dat  
; Z - returned as the Z data from gt_xyz.dat  
; Xf - returned as the X Fault data from gt_fault.dat  
; Yf - returned as the Y Fault data from gt_fault.dat  
; Xc - returned as the X Crease data  
; (for this example, same as the X Fault data)
```

```

        ;   Yc - returned as the Y Crease data
        ;   (for this example, same as the Y Fault data)
        ;
X = FLTARR(26)
Y = FLTARR(26)
Z = FLTARR(26)
Xf = FLTARR(60)
Yf = FLTARR(60)
T1 = 0.0
T2 = 0.0
T3 = 0.0
IF !Version.platform EQ 'vms' THEN BEGIN
    gtgrid_path = extract_path(!Option_path, "GTGRID")
    OPENR, Unit, STRMID(gtgrid_path, 0, STRLEN(gtgrid_path)-1)+$
        '.data\gt_fault.dat', /Get_lun
ENDIF ELSE BEGIN
    gtgrid_path = extract_path(!Option_path, "gtgrid")
    OPENR, Unit, gtgrid_path+'/data/gt_fault.dat', /Get_lun
ENDELSE
FOR I = 0, N_elements(Xf)-1 DO BEGIN
    READF, Unit, T1, T2
    Xf(I) = FLOAT(T1)
    Yf(I) = FLOAT(T2)
ENDFOR
FREE_LUN, Unit
Xc=Xf
Yc=Yf
;
IF !Version.platform EQ 'vms' THEN $
    OPENR, Unit, STRMID(gtgrid_path, 0, STRLEN(gtgrid_path)-1)+$
        '.data\gt_xyz.dat', /Get_lun $
    ELSE $
        OPENR, Unit, gtgrid_path+'/data/gt_xyz.dat', /Get_lun
FOR I = 0, N_elements(X)-1 DO BEGIN

```

```

        READF, Unit, T1, T2, T3
        X(I) = FLOAT(T1)
        Y(I) = FLOAT(T2)
        Z(I) = FLOAT(T3)
    ENDFOR
FREE_LUN, Unit
END

```

GT_CON_DATA File Listing

On a UNIX system, this file is located in the directory:

```

$VNI_DIR/gtgrid-2_0/lib
pro gt_con_data, x, y, z, xf, yf
;
x = fltarr(626)
y = fltarr(626)
z = fltarr(626)
xf = fltarr(69)
yf = fltarr(69)
t1 = 0.0
t2 = 0.0
t3 = 0.0
IF !Version.platform EQ 'vms' THEN BEGIN
    gtgrid_path = extract_path(!Option_path, "GTGRID")
    openr, unit, STRMID(gtgrid_path, 0, STRLEN(gtgrid_path)-1)+$
        '.data]gt_faults.dat', /get_lun
ENDIF ELSE BEGIN
    gtgrid_path = extract_path(!Option_path, "gtgrid")
    openr, unit, gtgrid_path+'/data/gt_faults.dat', /get_lun
ENDELSE
for i = 0, n_elements(xf)-1 do begin
    readf, unit, t1, t2
    xf(i) = float(t1)

```

```
        yf(i) = float(t2)
    endfor
    free_lun, unit
;
IF !Version.platform EQ 'vms' THEN $
    openr, unit, STRMID(gtgrid_path, 0, STRLEN(gtgrid_path)-1)+$
        '.data]gt_contours.dat', /get_lun $
    ELSE $
        openr, unit, gtgrid_path+'/data/gt_contours.dat', /get_lun
for i = 0, n_elements(x)-1 do begin
    readf, unit, t1, t2, t3
    x(i) = float(t1)
    y(i) = float(t2)
    z(i) = float(t3)
endfor
free_lun, unit
end
```


Error Messages

This chapter discusses error messages generated by the PV-WAVE:GTGRID routines. Following each message is a listing of the command or commands that caused the error, and finally a brief explanation of how to correct the error.

All PV-WAVE:GTGRID errors return the scalar value 0.0.

GTGRID Error Messages

Error Message:

```
%GTGRID: Argument XVEC must be a FLOAT or a DOUBLE array.
```

Commands Entered that Produced the Error:

```
WAVE> X = 1  
WAVE> Zz = GTGRID(X,Y,Z)
```

Response:

A solution to this error is to make *X* a float array. For example:

```
X = fltarr(5)  
WAVE> Zz = GTGRID(X,Y,Z)
```

Error Message:

```
%GTGRID: Direct method is limited to vectors with less than
100000 points.
```

Commands Entered that Produced the Error:

```
WAVE> X = FLTARR(110000)
WAVE> Y=X & Z=X
WAVE> Zz=GTGRID(X,Y,Z, Method='direct')
```

Response:

The direct method can only be used on data that contains less than 100000 points. Either change methods or reduce the number of points.

Error Message:

```
%GTGRID: Direct Method does NOT support fault OR crease
processing.
```

Commands Entered that Produced the Error:

```
WAVE> Zz=GTGRID(X, Y, Z, $
Method='direct', Xfaults=Xf)
```

Response:

The direct method can not be used with fault or crease data. Use another method if faults or creases need to be considered. Remove the *Xfaults* keyword if you really want to use the direct method.

Error Message:

```
%GTGRID: Invalid gridding method specified: BOGUS
```

Commands Entered that Produced the Error:

```
WAVE> Zz = GTGRID(X, Y, Z, Method='bogus')
```

Response:

Valid methods include: *Direct*, *Scatter*, *Cluster*, and *Weighted*. Any other string, including an empty string, will result in an error.

Error Message:

%GTGRID: Both XFAULTS and YFAULTS keywords must be supplied for fault processing.

Commands Entered that Produced the Error:

```
WAVE> Zz=GTGRID(X,Y,Z, Xfaults=Xf)
```

or

```
WAVE> Zz=GTGRID(X,Y,Z, Yfaults=Yf)
```

Response:

Both *Xfaults* and *Yfaults* keywords must be present for fault processing.

Error Message:

%GTGRID: Maximum number of data points exceeded.

%GTGRID: Reduce the number of data points or size of output grid.

Commands Entered that Produced the Error:

```
WAVE> X = FLTARR(1000000)
```

```
WAVE> Y = X & Z = X
```

```
WAVE> Zz=GTGRID(X, Y, Z)
```

Response:

Two million (2,000,000) is the maximum number of data points and grid nodes. GTGRID uses the following equation to determine whether the number of maximum points has been exceeded:

$$N_Elements(XVEC) + N_Elements(YVEC) + N_Elements(ZVEC) + (NX * NY) < 2000000$$

Error Message:

%GTGRID: The X clipping maximum must be greater than the minimum.

Commands Entered that Produced the Error:

```
WAVE> Zz=GTGRID(X,Y,Z, Xmin=1000,$  
Xmax=100)
```

Response:

The minimum value must be less than the maximum value or an error will occur. Change *Xmin* to less than 100 or increase *Xmax* to greater than 1000. This also applies to the *Y* clipping planes.

Error Message:

%GTGRID: The X and Y FAULTS arrays must be the same length.

Commands Entered that Produced the Error:

```
WAVE> Yf = findgen(60)  
WAVE> Xfbad = findgen(59)  
WAVE> ZZ = GTGRID(X, Y, Z, Xfault=Xfbad, $  
Yfault=Yf)
```

Response:

The number of *X* faults must be equal to the number of *Y* faults for fault processing.

Error Message:

```
% GTGRID: ERROR - No input points found in [xmin, xmax].
```

Commands Entered that Produced the Error:

```
WAVE> gt_get_data, X, Y, Z  
WAVE> ZZ = GTGRID(X, Y, Z, xmin=0.0, $  
xmax=1000.0)
```

Response:

Points in X between $Xmin$ and $Xmax$, inclusive, must exist for a grid to be produced. Change $Xmax$ so that the range includes a subset of X .

Error Message:

```
% GTGRID: Interpolation distance (interp_dist) must be GT 0
```

Commands Entered that Produced the Error:

```
WAVE> gt_get_data, X, Y, Z  
WAVE> ZZ = GTGRID(X, Y, Z, $  
interp_dist = -10.0)
```

Response:

The interpolation distance must be positive and nonzero. Change the value to 10.0 or use the default.

Error Message:

```
% GTGRID: XSPACING and YSPACING must be GT 0: xspacing = -  
100.250 yspacing = 101.895
```

Commands Entered that Produced the Error:

```
WAVE> gt_get_data, X, Y, Z  
WAVE> ZZ = GTGRID(X, Y, Z, xspacing = -100.25)
```

Response:

The spacing between grid nodes must be positive and nonzero. Change the value to 100.25 or use the default.

Error Message:

```
% GTGRID: Neighbors must be GE 0, Default == 5
```

Commands Entered that Produced the Error:

```
WAVE> gt_get_data, X, Y, Z  
WAVE> ZZ = GTGRID(X, Y, Z, neighbors = -1)
```

Response:

The number of octants required to have neighbors for node calculation must be positive. Change the value to 0 or use the default of 5.

Error Message:

```
% GTGRID: Averaging distance (dist_avg) must be GT 0
```

Commands Entered that Produced the Error:

```
WAVE> gt_get_data, X, Y, Z  
WAVE> ZZ = GTGRID(X, Y, Z, dist_avg = -5.0)
```

Response:

The distance over which to average input points before using them to compute the grid nodes must be positive and nonzero. Set the *Dist_avg* to 5.0 or use the default.

Error Message:

```
**** WARNING! CREASES are no longer supported in GTGRID.
****
```

Commands Entered that Produced the Error:

```
WAVE> gt_get_data, X, Y, Z, XF, YF, XC, YC
WAVE> ZZ = GTGRID(X, Y, Z, xcrease=XC, $
ycrease=YC)
```

Response:

Creases are not supported in PV-WAVE:GTGRID 2.x. GTGRID function calls that use creases will run without error; however, the creases are ignored. Avoid using *Xcrease* or *Ycrease* keywords in PV-WAVE:GTGRID 2.x calls.

GTGETGRID Error Messages

Error Message:

```
% GTGETGRID: nx must be GT 0.
```

Commands Entered that Produced the Error:

```
WAVE> ZZ = GTGETGRID(surf_id, nx = -1, ny = 10)
```

Response:

The requested grid size must be valid array dimensions.

GTGETVALUE Error Messages

Error Message:

```
% GTGETVALUE: xval and yval must have the same number of
elements.
```

Commands Entered that Produced the Error:

```
WAVE> x1 = findgen(100)+3000.0
```

```
WAVE> y1 = findgen(10)+2000.0
WAVE> ZVAL = GTGETVALUE(surf_id, x1, y1)
```

Response:

The arrays of x and y coordinates must be of equal size. Pass GTGETVALUE a portion of the x_I array, or extend the y_I array to the length of x_I .

GTINSTALLSURF Error Messages

Error Message:

```
% GTINSTALLSURF: Must provide a 2D grid as input.
```

Commands Entered that Produced the Error:

```
WAVE> badZz = 1
WAVE> mysurf=GTINSTALLSURF(badZz)
```

Response:

The installed grid must be two-dimensional.

Error Message:

```
% GTINSTALLSURF: Maximum number of data points exceeded.
```

Commands Entered that Produced the Error:

```
WAVE> mysurf=GTINSTALLSURF(Zz, $
numpts=3000000L)
```

Response:

Two million (2,000,000) is the maximum number of data points and grid nodes. GTGRID uses the following equation to determine whether the number of maximum points has been exceeded:

$$N_Elements(XVEC) + N_Elements(YVEC) + N_Elements(ZVEC) + (NX * NY) < 2000000$$

Error Message:

```
% GTINSTALLSURF: numpts must be GE 0.
```

Commands Entered that Produced the Error:

```
WAVE> mysurf=GTINSTALLSURF(Zz, numpts=0L)
```

Response:

A grid must be associated with a nonzero number of points. As a solution you can specify the largest grid dimension as the number of points.

Error Message:

```
% GTINSTALLSURF: nx must be GE 0.
```

Commands Entered that Produced the Error:

```
WAVE> mysurf=GTINSTALLSURF(Zz, nx=-1, $  
numpts=26)
```

Response:

The dimensions of the installed grid must be nonzero and positive. nx and ny will default to the dimensions of the grid, Zz.

Error Message:

```
% GTINSTALLSURF: Must supply xvec, xmin and xmax, or  
xspacing.
```

Commands Entered that Produced the Error:

```
WAVE> mysurf=GTINSTALLSURF(Zz, yvec=yvec)
```

Response:

GTINSTALLSURF must be able to determine the appropriate x , y origin and node spacing for the given grid. That information can only be computed from an x vector ($Xvec$), the minimum and maximum x values. Otherwise, the x -spacing must be explicitly defined. The y -spacing must also be determined.

Error Message:

% GTINSTALLSURF: Both XFAULTS and YFAULTS keywords must be supplied for fault processing.

Commands Entered that Produced the Error:

```
WAVE>mysurf=GTINSTALLSURF(Zz, xfaults=Xf, $
numpts=26, xvec=X, yvec=Y)
```

Response:

Both x and y faults must be provided for fault installation.

Error Message:

% GTINSTALLSURF: The X and Y FAULTS arrays must be the same length.

Commands Entered that Produced the Error:

```
WAVE> gt_get_data, X, Y, Z, XF, YF
WAVE> XFbad = 1
WAVE> mysurf=GTINSTALLSURF(Zz, $
xfaults=XFbad, yfaults=YF, numpts=26)
```

Response:

The number of x faults must be equal to the number of y faults for fault installation.

Error Message:

```
% GTINSTALLSURF: xspacing must be GE 0.
```

Commands Entered that Produced the Error:

```
WAVE>gt_get_data,X,Y,Z  
WAVE>mysurf=GTINSTALLSURF(Zz, $  
xspacing=-100.0, yspacing=120.0, $  
numpts=26, xvec=X, yvec=Y, zvec=Z)
```

Response:

The defined spacing between nodes in the installed grid must be positive and nonzero.

Error Message:

```
% GTINSTALLSURF: yspacing must be GE 0.
```

Commands Entered that Produced the Error:

```
WAVE>gt_get_data, X, Y, Z  
WAVE> ymin=min(Y, max=yymax) & $  
xmin=min(X, max=xmax) & zmin=MIN(Z)  
WAVE>mysurf=GTINSTALLSURF(Zz, $  
xspacing=100.0, ymin=yymax, ymax=ymin, $  
xorg=xmin, yorg=ymin, zmin=zmin, numpts=26)
```

Response:

Unless explicitly defined, the defined spacing between nodes in the installed grid is determined by the formula:

```
xspacing = FLOAT((xmax-xmin)/((FLOAT(nx)-1.)))  
yspacing = FLOAT((ymax-ymin)/((FLOAT(ny)-1.)))
```

Make sure that $ymin < ymax$ and $xmin < xmax$.

Error Message:

% GTINSTALLSURF: Must supply *xvec*, *xmin*, or *xorg*.

Commands Entered that Produced the Error:

```
WAVE>mysurf=GTINSTALLSURF(Zz, $  
xspacing=100.0, yspacing=120.0, numpts=26)
```

Response:

GTINSTALLSURF must determine the origin of the grid in order to install it. This can only be determined from the original *x* vector from which the grid was formed or the minimum *x* value. Otherwise, the origin must be explicitly defined. The keywords *Yvec*, *Ymin*, or *Yorg* must also be defined.

Error Message:

% GTINSTALLSURF: Must supply *zvec*, *zmin* and *zmax*, or *nulval*.

Commands Entered that Produced the Error:

```
WAVE> mysurf=GTINSTALLSURF(Zz, $  
xspacing=100.0, yspacing=120.0, xorg=xmin, $  
yorg=ymin, zmin=zmin, numpts=26)
```

Response:

GTINSTALLSURF must determine the *Nulval*, the value meaning an undefined point, for the installed grid. Unless explicitly defined, the *Nulval* is determined as follows:

```
nulval = FLOAT(zmin) - (FLOAT(zmax) - FLOAT(zmin))
```

If the *z* vector is given by the user, it is used to determine *Zmin* and *Zmax*. Otherwise, *Zmin* and *Zmax* must both be defined.

GTRESTORESUF and GTSAVESURF Error Messages

Error Message:

```
% GTRESTORESUF: Filename cannot end with /
```

Commands Entered that Produced the Error:

```
WAVE> surf_id = GTRESTORESUF('mysurf.srf/')
```

Response:

A filename cannot end in the slash (/) character. This causes GTRESTORESUF and GTSAVESURF to improperly treat the entire filename as a directory.

Error Message:

```
% GTRESTORESUF: Filenames greater than 40 characters not supported.
```

Commands Entered that Produced the Error:

```
WAVE> surf_id = $  
GTRESTORESUF('averyveryveryreallyreallylongfilenamehere')
```

Response:

Filenames, not including the path specification, are limited to 40 characters in length.

GTGRID Index

A

application, example of 6
average distance between points 15

B

bounding polygon
retrieving 10
specifying 15

C

clipping of input data 18
cluster method 21

D

demonstration program 6
destroy surface 9
Digital UNIX restrictions 6
direct method 20

E

error messages B-1
example GTGRID application 6
examples A-1
exiting PV-WAVE
GTGRID 2

F

fault
retrieving 11
fault data

processing 5
showing 17, 23
free memory 9

G

Geophysical Techniques, Inc. 1
GTDESTROYSURF procedure 9
GTGETBOUNDS function 10
GTGETFAULTS function 11
GTGETGRID function 12
GTGETSURFS function 13
GTGETVALUE function 14
GTGRID
definition of 1
general procedure for using 3
GTGRID function 15
GTGRID_STARTUP 2
GTGRID_UNLOAD 2
GTINSTALLSURF function 22
GTRESTORESURF function 25
GTSAVESURF procedure 26

M

memory space, releasing 9
method
cluster 21
direct 20
keywords 19
scatter 21
selecting a 4
specifying a 15
weighted 21

N

- new features 2
- node, definition of 4
- null values, specifying 16

P

- polygon, bounding 10
- primary estimates of Z values 4
- process size, managing 7

R

- radial search
 - for clustered or linear data 21
 - for scattered points 21

S

- scatter method 21
- secondary estimates of Z values 4
- smoothing
 - specifying 16
- smoothing, general discussion of 6
- starting PV-WAVE:GTGRID 2
- stopping PV-WAVE:GTGRID 2
- surface
 - creating 22
 - grid matrix 12
 - ID description 10
 - ID, returning a 16
 - list active 13
 - preserving information about 16
 - restoring 25
 - saving 26
 - value of point on 14

U

- unloading an option 10

W

- weighted
 - algorithm 21
 - method 21

Z

- Z values
 - clipping 18
 - primary estimates of 4
 - secondary estimates of 4