IDL®

Getting Started with IDL

RESEARCH SYSTEMS
A Kodak Company

IDL Version 5.6
October, 2002 Edition

1002IDL56GS

Acknowledgments

IDL® is a registered trademark and ION™, ION Script™, ION Java™, are trademarks of Research Systems Inc., registered in the United States Patent and Trademark Office, for the computer program described herein.

Numerical Recipes™ is a trademark of Numerical Recipes Software. Numerical Recipes routines are used by permission.

GRG2™ is a trademark of Windward Technologies, Inc. The GRG2 software for nonlinear optimization is used by permission.

NCSA Hierarchical Data Format (HDF) Software Library and Utilities
Copyright 1988-2001 The Board of Trustees of the University of Illinois
All rights reserved.

NCSA HDF5 (Hierarchical Data Format 5) Software Library and Utilities
Copyright 1998, 1999, 2000, 2001, 2002 by the Board of Trustees of the University of Illinois. All rights reserved.

CDF Library
Copyright © 1999
National Space Science Data Center
NASA/Goddard Space Flight Center

NetCDF Library
Copyright © 1993-1996 University Corporation for Atmospheric Research/Unidata

HDF EOS Library
Copyright © 1996 Hughes and Applied Research Corporation

This software is based in part on the work of the Independent JPEG Group.

Portions of this software are copyrighted by INTERSOLV, Inc., 1991-1998.

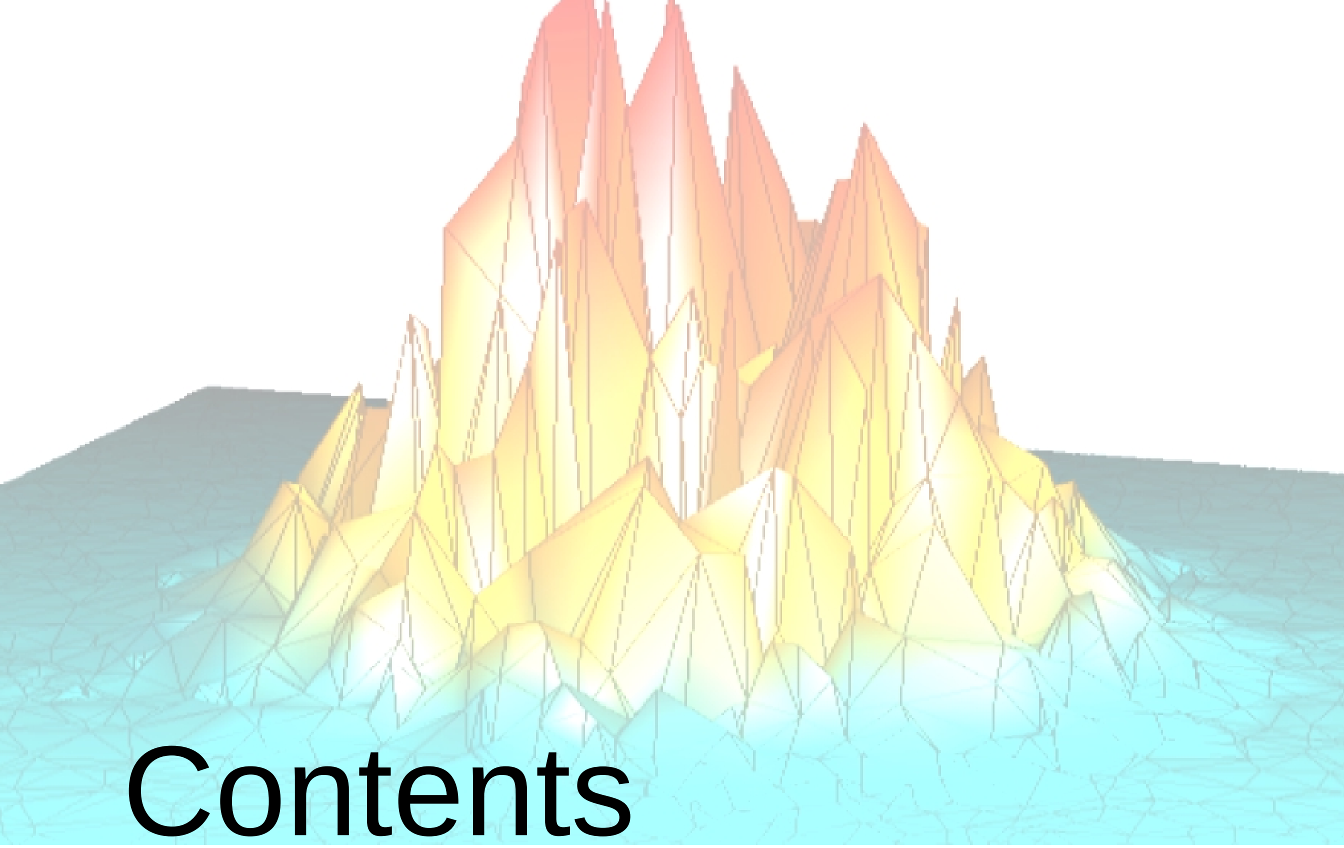
Use of this software for providing LZW capability for any purpose is not authorized unless user first enters into a license agreement with Unisys under U.S. Patent No. 4,558,302 and foreign counterparts. For information concerning licensing, please contact: Unisys Corporation, Welch Licensing Department - C1SW19, Township Line & Union Meeting Roads, P.O. Box 500, Blue Bell, PA 19424.

Portions of this computer program are copyright © 1995-1999 LizardTech, Inc. All rights reserved. MrSID is protected by U.S. Patent No. 5,710,835. Foreign Patents Pending.

This product includes software developed by the Apache Software Foundation (http://www.apache.org/)

IDL Wavelet Toolkit Copyright © 2002 Christopher Torrence.

Other trademarks and registered trademarks are the property of the respective trademark holders.



Contents

Chapter 1:
The Power of IDL 9

Using this Manual 12

Chapter 2:
The IDL Development Environment 13

IDL's Development Environment 14

Starting IDL 15

The IDL Interfaces 16

The IDL Development Environment Layout 17

Quitting IDL 20

More Information on the IDLDE 21

Chapter 3:
Reading and Writing Data 23

IDL and Reading and Writing Data 24

IDL Supported Formats 25
    Image Formats 25
    Scientific Data Formats 25
    Other Formats 25
Importing Data from an ASCII File 26
Reading and Writing Binary Data 31
Saving a Template 35
Reading and Writing Images 36
More Information on IDL and Input/Output 38

Chapter 4:
2-D Plotting 39
IDL and 2-D Plotting 40
Simple Plotting 41
    Using OPLOT 42
    Printing a Plot 43
Plotting with Data Sets 44
Other Plotting Capabilities 45
Using LIVE_PLOT 46
More Information on 2-D Plotting 48

Chapter 5:
Signal Processing 49
IDL and Signal Processing 50
Creating a Data Set 51
Signal Processing with SMOOTH 54
Frequency Domain Filtering 55
Displaying the Results 58
More Information on Signal Processing 60

Chapter 6:
Image Processing 61
IDL and Image Processing 62
Reading an Image 63
Displaying an Image 64
Resizing an Image 66
Resizing a Graphics Window 67

Contrast Enhancement 68
Thresholding 68
Scaling Pixel Values 71
Smoothing and Sharpening 74
Unsharp Masking 75
Sharpening Images with Differentiation 76
Other Image Manipulations 78
Rotating an Image 79
Extracting Profiles 80
Using LIVE_IMAGE 81
More Information on Image Processing 82

Chapter 7:
Surface and Contour Plotting 83

IDL and Surface and Contour Plotting 84
Reading a Dataset to Plot 85
Displaying a Surface 86
Displaying a Shaded Surface 88
Displaying a Contour 91
Plotting with SHOW3 96
Using LIVE_SURFACE for Plotting 97
More Information on 3-D Plotting 98

Chapter 8:
Volume Visualization 99

IDL and Volume Visualization 100
Reading in a Dataset for Visualization 101
3-D Transformations 102
Visualizing an Iso-Surface 104
Making Slices with the IDL Slicer 106
Displaying a Surface with the Slicer 108
Dismiss the Slicer and Volume Windows 109
More Information on Volume Visualization 110

Chapter 9:
Mapping 111

IDL and Mapping 112

Drawing Map Projections 113
Drawing an Orthographic Projection 115
Plotting a Portion of the Globe 116
Plotting Data on Maps 117
Reading Latitudes and Longitudes 119
Plotting Contours Over Maps 120
Warping Images to Maps 122
More Information on Mapping 127

Chapter 10:
Plotting Irregularly-Gridded Data 129

IDL and Plotting Irregularly-Gridded Data 130
Creating a Dataset 131
The TRIANGULATE Procedure 133
Plotting the Results with TRIGRID 135
More Information on Gridding 137

Chapter 11:
Animation 139

IDL and Animation 140
Animating a Series of Images 141
Displaying an Animation as a Wire Mesh 143
Animation with XINTERANIMATE 146
Cleaning Up the Animation Windows 148
More Information on Animation 149

Chapter 12:
Programming in IDL 151

IDL and Programming 152
Programming Capabilities in the IDLDE 153
- Built-In Editor 153
- Types of IDL Programs 153
- Compound Statements 154
- Conditional Statements 154
- Loop Statements 154
- Jump Statements 154

Executing a Simple IDL Example Program 155

Saving, Compiling and Running your Program 156
Debugging Tools in IDL 158
Breakpoints 158
Variable Watch window 158
The IDL Code Profiler 159
Using IDL Projects 160
Access to all Files in Your Application 160
Working with Files in Your Project 161
Compiling and Running Your Application 161
Building Distributions 161
Exporting Your Applications 161
More Information on IDL Programming 162

Chapter 13:
Manipulating Data 163

IDL and Manipulating Data 164
IDL Array Routines 165
Array Creation Routines 165
Array Manipulation Routines 166
Array and Image Processing Routines 166
Array Processing Capabilities 168
More Information on Manipulating Data 171

Chapter 14:
Using the IDL GUIBuilder 173

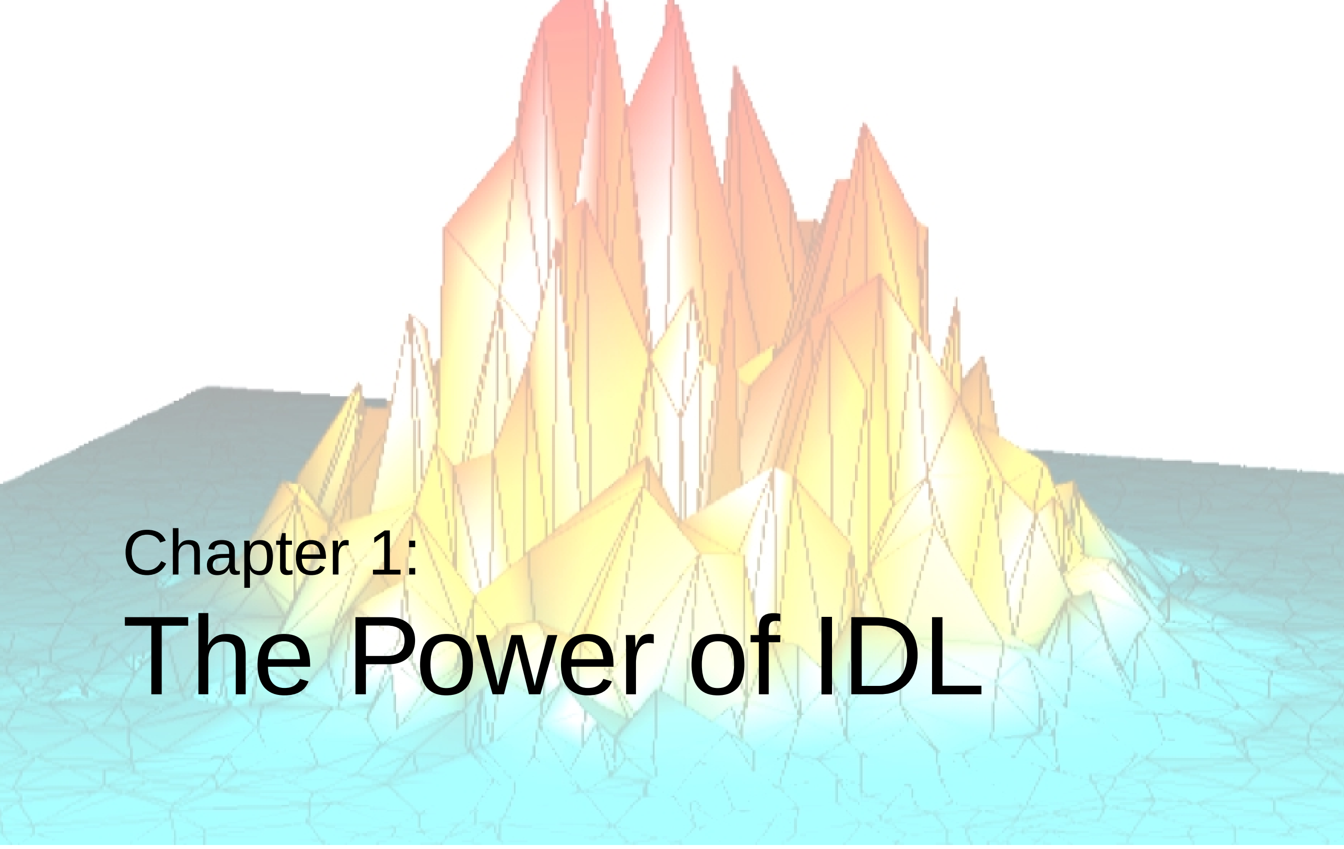
What is the IDL GUIBuilder? 174
Using the IDL GUIBuilder 174
IDL GUIBuilder Tools 176
Using the IDL GUIBuilder Toolbar 176
Creating an Example Application 177
Defining Menus for the Top-Level Base 177
Running the Application in Test Mode 181
Generating the IDL Code 181
Handling the Open File Event 182
Handling the Exit Event 183
Handling the Load Color Table Event 184
Handling the Smooth Event 184

Compiling and Running the Example Application 185
Widget Types 187
Widget Properties 188
More Information on the IDL GUIBuilder 189

Chapter 15:
Where to Go From Here 191

Learning More about IDL 192
IDL Documentation Set 192
Online Manuals 194
Online Help 195
IDL Demo Applications and Examples 196
Contacting RSI 197
Address 197
Phone 197
Fax 197
E-mail 197
World Wide Web 197

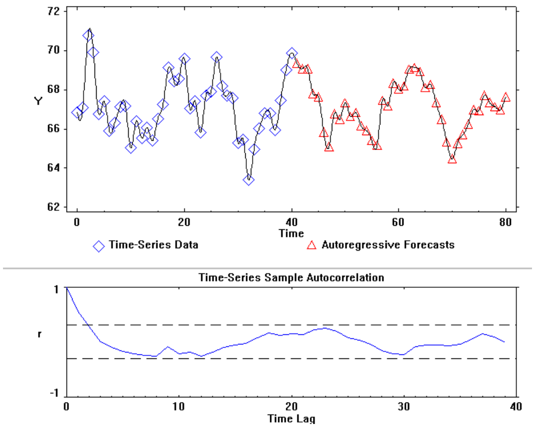
Index 199



Chapter 1: The Power of IDL

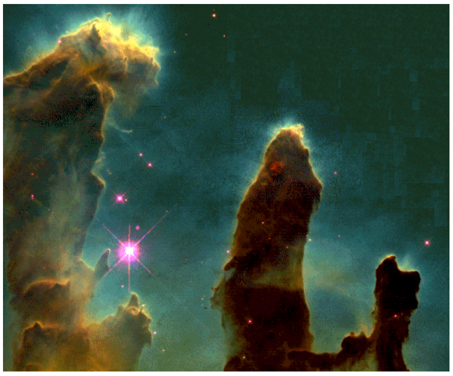
IDL, the *Interactive Data Language*, is the ideal software for data analysis, visualization, and cross-platform application development. IDL integrates a powerful, array-oriented language with numerous mathematical analysis and graphical display techniques, thus giving you incredible flexibility. A few lines of IDL can do the job of hundreds of lines of C or Fortran, without losing flexibility or performance. A fourth-generation language, IDL is radically more compact than C or Fortran. Using IDL, tasks that require days or weeks of programming with traditional languages can be accomplished in hours. Users can explore data interactively using IDL commands and then create complete applications by writing IDL programs.

Create Data in IDL - Use IDL to create data, using a complete, structured language that can be used interactively and on sophisticated functions, procedures, and applications.



Use the IDLDE to create Applications - Use the IDLDE (IDL Development Environment) to compile and execute commands immediately. It also includes built-in editing and debugging tools that provide instant feedback and "hands-on" interaction.

Read and Write Data in IDL - Use IDL to read and write almost any kind of data. Support is provided for common image standards and scientific data formats. If you have data, you can read it into IDL!

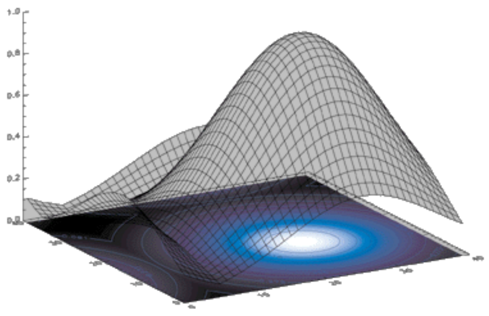


Create Plots in IDL - IDL includes many 2-D Plotting techniques, to observe the results of your computations immediately.

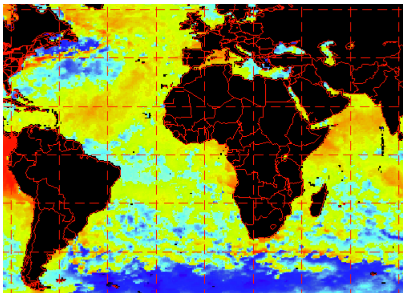
Signal Processing in IDL - Use IDL Signal Processing techniques to process a variety of 1-D signals, from traditional filtering and transform operations to statistical methods such as prediction analysis.

Surface and Contour Plots - Use IDL Surface and Contour Plotting techniques to display any 2-D dataset as surface.

Image Processing in IDL - Use IDL Image Processing techniques to filter out noise and to highlight true data characteristics and expose anomalies.



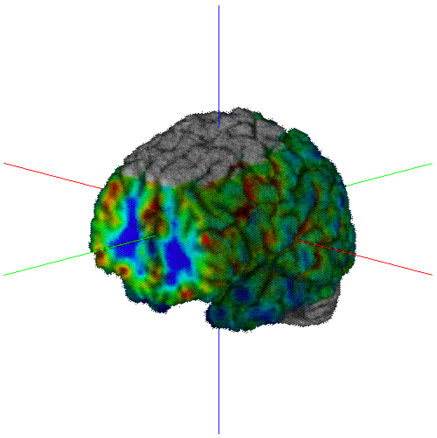
Volume Visualization in IDL - Use IDL Volume Visualization functionality to visualize 3-D volume datasets and to display a shaded surface representation of a constant-density surface (also called an iso-surface).



Mapping Capabilities in IDL - Use IDL Mapping techniques to plot data over different projections of the globe.

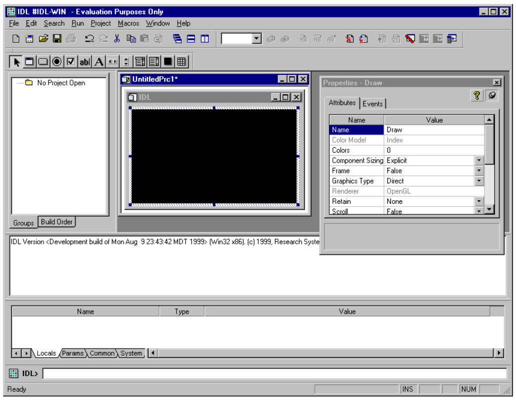
Irregularly-Sampled Data - Use IDL to easily fit irregularly-sampled data to a regular grid. This regularly-gridded data can then be sent to IDL's plotting routines.

Animation in IDL - Use IDL for Animation tasks to visualize your data dynamically and to create an array of images and play them back as an animated sequence.



Create Applications in IDL - Use IDL to write sophisticated programs and applications using a complete set of program-control statements.

IDL GUIBuilder - Use the IDL GUIBuilder to interactively create user interfaces and then generate the IDL source code that defines that interface.


IDL #IDL-WIN - Evaluation Purposes Only
File Edit Search Run Project Macros Window Help
No Project Open
UntitledPrc1*
IDL
Properties - Draw
Attributes Events
Groups Build Order
Locals Params Common System
IDL>
Ready
INS NUM

Name	Value
Name	Draw
Color Model	Index
Colors	0
Component Sizing	Explicit
Frame	False
Graphics Type	Direct
Renderer	OpenGL
Retain	None
Scroll	False

```
IDL Version <Development build of Mon Aug  9 23:43:42 MDT 1999> (Win32 x86). (c) 1999, Research Syste
```

Name	Type	Value

Using this Manual

The chapters included in this manual provide a “hands-on” way to learn basic IDL concepts and techniques. *Getting Started with IDL* demonstrates a number of common IDL applications: reading and writing data, 2-D plotting, signal processing, surface and contour plotting, image processing, volume visualization, mapping, plotting irregularly-gridded data, animation, programming in IDL, manipulating data, IDL Toolkits, and use of IDL’s GUIBuilder. Each section introduces basic IDL concepts and highlights some of the commonly used IDL commands.

You don’t have to read all of the descriptive passages that accompany each chapter. Simply enter the IDL commands shown in `courier` type at the IDL Command Input Line (the “IDL>” prompt) and observe the results. Unless otherwise noted, each line shown is a complete IDL command (press RETURN after typing each command). If you want more information about a specific command, you can read the explanations.

Each chapter functions similarly to a tutorial and is a demonstration of a particular IDL feature. It is recommended that you walk through each short, interactive chapter to preserve continuity, since many commands rely upon previous commands. Each chapter assumes the most basic level of IDL experience.

Note

The examples and graphics in this manual have been captured using the Windows platform. Where needed, explanations have been provided for use of the examples on UNIX or Macintosh platforms.

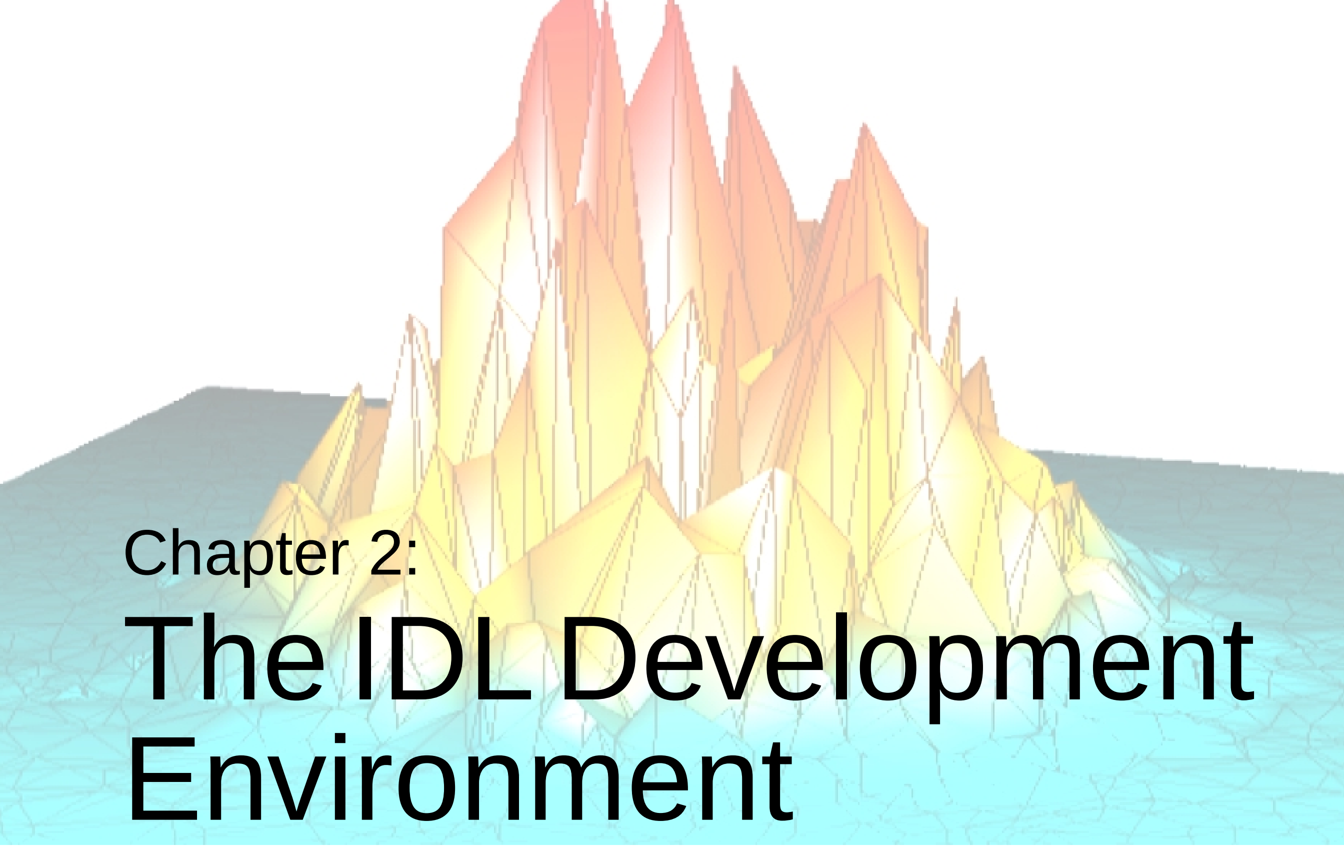
Note

The dollar sign ($) at the end of the first line is the IDL continuation character. It allows you to enter long IDL commands as multiple lines.

Note

To simplify obtaining useful results from the examples in this manual, create a bitmap buffer for your graphic windows and use a maximum of 256 colors by entering the following command at the IDL command prompt:

```
DEVICE, RETAIN=2, DECOMPOSED=0
```



Chapter 2: The IDL Development Environment

This chapter introduces the IDL Development Environment and its capabilities.

IDL's Development Environment 14
Starting IDL 15
The IDL Interfaces 16
Quitting IDL 20
More Information on the IDLDE 21

IDL’s Development Environment

IDL has a convenient multiple-document interface called the IDL Development Environment (sometimes also referred to as the IDLDE) that includes built-in editing and debugging tools. The IDL Development Environment is available for use on the Windows, Macintosh, and Motif (UNIX) platforms. This chapter briefly addresses starting IDL, quitting IDL, and describes the components of the IDL Development Environment.

Starting IDL

To run IDL, follow the instructions below.

1. Install and License IDL. For more information, see the IDL installation guide for your platform.
2. Start IDL.

   For Windows, click the Windows Start button, and select Programs → Research Systems IDL 5.6 → IDL.

   For Macintosh:

   A. Navigate to the OroborOSX installation folder.

   B. Double-click the **OroborOSX** icon. OroborOSX launches XDarwin and displays a UNIX X-Windows command line in an OS X window.

   C. Enter `idlde` at the UNIX command prompt.

   For Motif, start IDL by entering the following at the % prompt:

   ```
   idlde
   ```

The IDL Interfaces

IDL's multiple-document interface is called the IDL Development Environment (IDLDE) and includes built-in editing and debugging tools.

Note

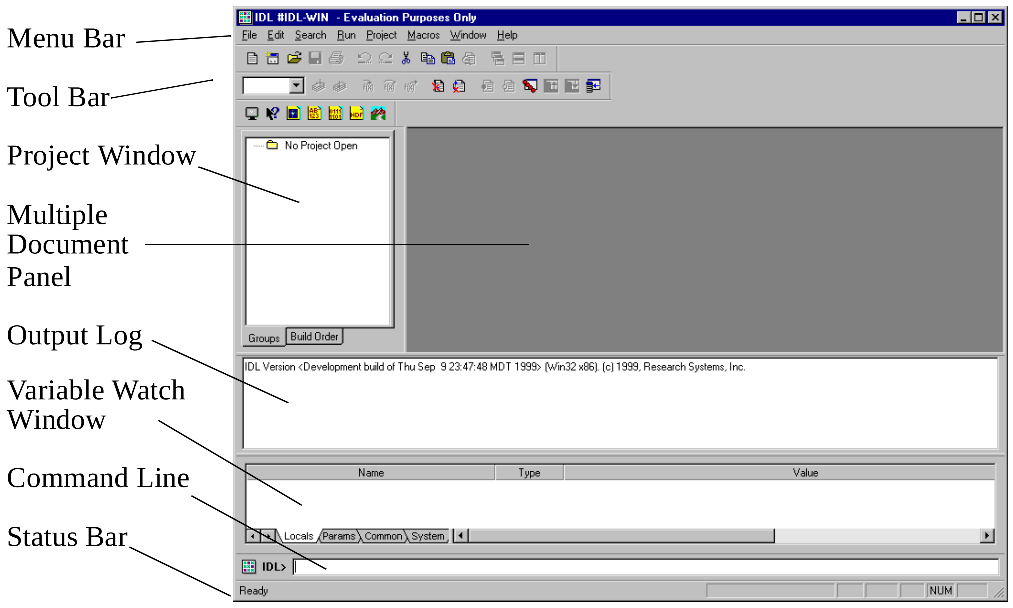
A command line interface is also available on UNIX and Macintosh platforms. For more information, see the *Using IDL* manual.

Note

All figures which are shown in this chapter are Windows environment figures but the IDLDE is very similar on each of the other platforms as well. Simply open the IDLDE on your own environment and follow along with the descriptions of IDLDE features.

The IDL Development Environment Layout

When you start IDL, the IDL Development Environment appears.



```
IDL Version <Development build of Thu Sep  9 23:47:48 MDT 1999> (Win32 x86). (c) 1999, Research Systems, Inc.
```

Figure 2-1: The IDL Development Environment for Windows

Menu Bar

The Menu Bar, located at the top of the main IDL window, is used to control various IDLDE features. When you select an option from the Menu Bar in the Development Environment, the Status Bar displays a brief description.

Tool Bar

There are three Tool Bars in the IDLDE: Standard, Run & Debug, and Macros. In addition, when you open an IDL GUIBuilder window, its associated Tool Bar is displayed. When you position the mouse pointer over a Tool Bar button, the Status Bar displays a brief description. If you click on a Tool Bar button which represents an IDL command, the IDL command issued is displayed in the Output Log.

Project Window

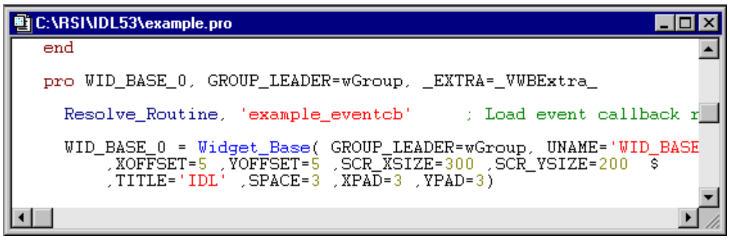
The Project Window displays information about the current Project you have open in the IDLDE. IDL Projects allow you to easily develop applications in IDL. You can manage, compile, run, and create distributions of all the files you will need to develop your IDL application. All of your application files can be organized so that they are easier to access and easier to export to other developers, colleagues, or users.

Multiple Document Panel

The top section of the main IDL window is where IDL Editor windows are displayed. The IDL Editor is where you create applications in IDL. To see the Multiple Document Panel at work, open the file examples.pro which can be found in the path:

```
rsi/idl56/examples/visual/examples.pro
```

Notice the color coding of commands, comments, and so on.



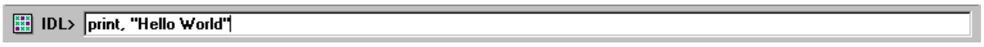
```
end

pro WID_BASE_0, GROUP_LEADER=wGroup, _EXTRA=_VWBExtra_

  Resolve_Routine, 'example_eventcb'     ; Load event callback r

  WID_BASE_0 = Widget_Base( GROUP_LEADER=wGroup, UNAME='WID_BASE
      ,XOFFSET=5 ,YOFFSET=5 ,SCR_XSIZE=300 ,SCR_YSIZE=200  $
      ,TITLE='IDL' ,SPACE=3 ,XPAD=3 ,YPAD=3)
```

Figure 2-2: Editor Window showing example.pro

Command Line

The Command Line is a single IDL prompt where you can enter IDL commands. If you click the right mouse button while positioned over the Command Input Line, a popup menu appears displaying the command history, with a default buffer of 10 entries and a maximum of 100 entries. IDL is an interpreted language and commands are therefore executed immediately at the command line. To see the IDL Command Line in action, enter the following in the Command Line at the IDL prompt and press Enter:

```
    print, 'Hello World!'
```

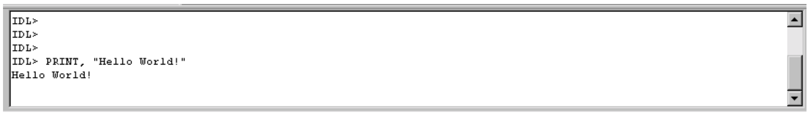


```
IDL> print, "Hello World"
```

Figure 2-3: Entering data at the IDL Command Line

Output Log

Output from IDL is displayed in the Output Log window, which appears by default when the IDLDE is first started (notice the result of our print command in the Output Log).



```
IDL>
IDL>
IDL>
IDL> PRINT, "Hello World!"
Hello World!
```

Figure 2-4: The IDL Output Log

Variable Watch Window

The Variable Watch window appears by default when you start the IDLDE. It keeps track of variables as they appear and change during program execution (tabs exist for viewing variables by type; Locals, Params, Common and System).

Status Bar

When you position the mouse pointer over a Control Panel button or select an option from a menu item in the IDLDE, the Status Bar displays a brief description.

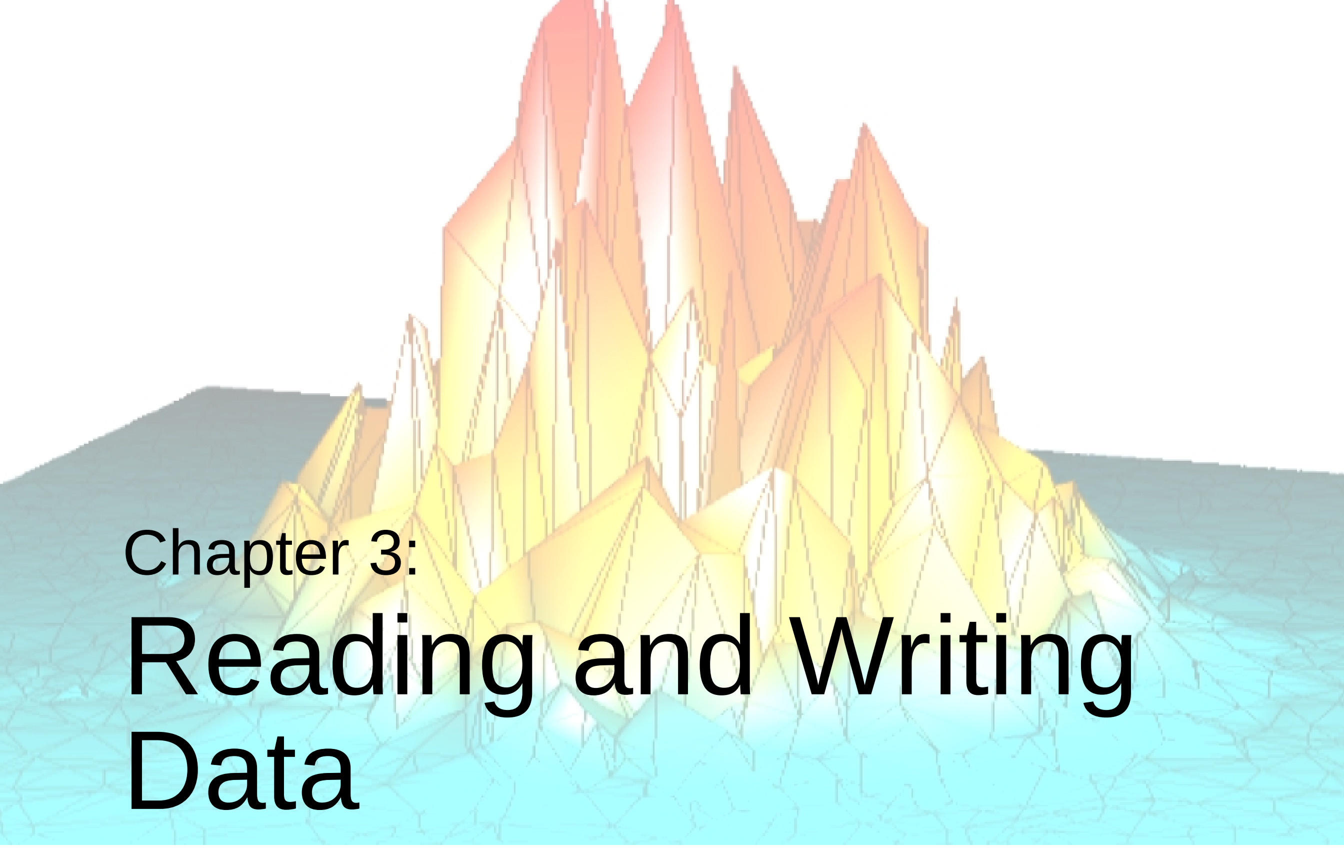
Quitting IDL

To quit the current IDL session and return to the operating system, select File → Exit in the IDL Development Environment. You can also type `EXIT` at the IDL Command Line:

```
IDL>EXIT
```

More Information on the IDLDE

This overview has acquainted you with the very basic layout and function of the IDL Development Environment. More in-depth information on working with IDL and the IDL Development Environment (IDLDE) can be found in *Using IDL*.



Chapter 3: Reading and Writing Data

This chapter introduces IDL's ability to read and write data.

IDL and Reading and Writing Data 24
IDL Supported Formats 25
Importing Data from an ASCII File 26
Saving a Template 35
Reading and Writing Binary Data 31
Reading and Writing Images 36
More Information on IDL and Input/Output 38

IDL and Reading and Writing Data

IDL’s flexible input and output capabilities allow you to read and write virtually any data format. When IDL reads a data file, bytes or characters in the file are converted to the appropriate data type (unless the file is binary, in which case no conversion takes place). Similarly, when writing data, the appropriate IDL variables are converted to the appropriate bytes or characters. In this chapter, you’ll import some existing data using IDL commands.

Note

To simplify obtaining useful results from the examples in this manual, create a bitmap buffer for your graphic windows and to use a maximum of 256 colors by entering the following command at the IDL command prompt:

```
DEVICE, RETAIN=2, DECOMPOSED=0
```

IDL Supported Formats

Image Formats

IDL includes routines for reading and writing many standard graphics file formats. These routines and the types of files they support are listed below. Documentation on these routines can be found in the *IDL Reference Guide*.

BMP	PNG
GEO TIFF	PPM
Interfile	SRF
JPEG	TIFF
NRIF	XWD
PICT	X11 Bitmap

Scientific Data Formats

There are four self-describing scientific data formats supported by IDL:

CDF (Common Data Format)	HDF (Hierarchical Data Format)
HDF-EOS (Earth Observing System extensions to HDF)	netCDF (Network Common Data Format)

Detailed documentation for each format can be found in the Scientific Data Formats manual.

Other Formats

ASCII	Binary
DICOM	WAV (Audio)
DXF	XDR (External Data Representation)

Importing Data from an ASCII File

One way of importing data in IDL is using the ASCII_TEMPLATE function in conjunction with the READ_ASCII function. To import an ASCII data file into IDL, you must first describe the format of the data using the ASCII_TEMPLATE function.

1. At the IDL Command Line, enter the following:

   ```
   PLOTTEMPLATE=ASCII_TEMPLATE()
   ```

   This command will guide you through assigning the description of the data to a variable named PLOTTEMPLATE. A dialog box appears, prompting you to select a file.

2. Select the file "plot.txt" located in the data directory:

   ```
   rsi-directory/examples/data/plot.txt
   ```

   Where *rsi-directory* is the installation directory for IDL.

Note

Another way to import ASCII data is to use the **Import ASCII File** toolbar button on the IDLDE toolbar. To use this feature simply click the button and the dialog will appear so that you may select `plot.txt`.

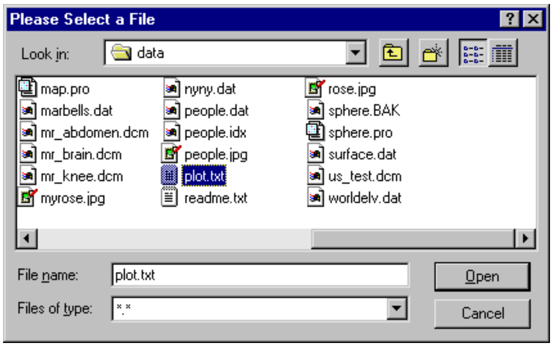
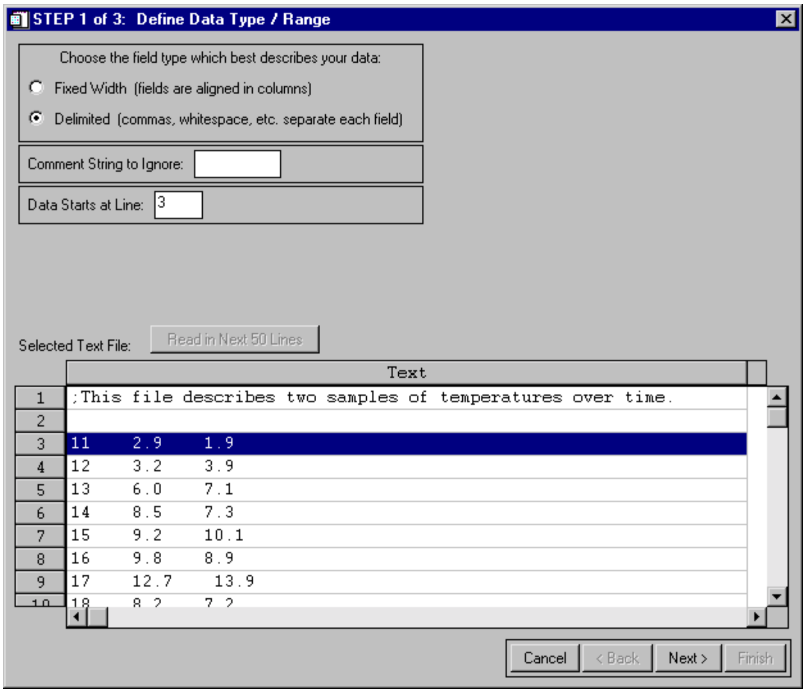


Figure 3-1: Selecting the plot.txt file

3. The **Define Data Type/Range** dialog appears:


STEP 1 of 3: Define Data Type / Range
Choose the field type which best describes your data:
Fixed Width (fields are aligned in columns)
Delimited (commas, whitespace, etc. separate each field)
Comment String to Ignore:
Data Starts at Line: 3
Selected Text File:
Read in Next 50 Lines
Text
Cancel
< Back
Next >
Finish

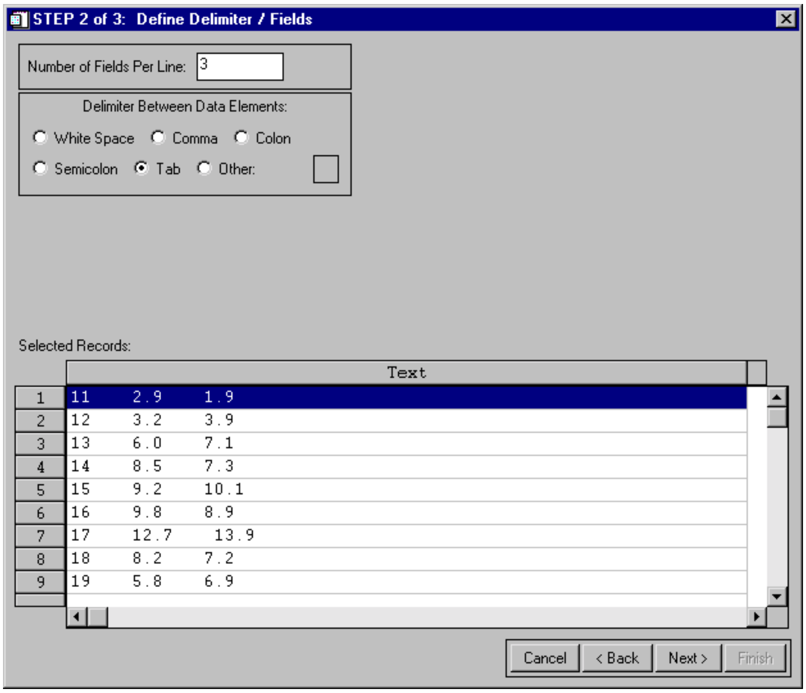
```
1  ;This file describes two samples of temperatures over time.
2
3  11     2.9     1.9
4  12     3.2     3.9
5  13     6.0     7.1
6  14     8.5     7.3
7  15     9.2     10.1
8  16     9.8     8.9
9  17     12.7     13.9
10 18     8.2     7.2
```

Figure 3-2: the ASCII_TEMPLATE dialog

4. First we will choose the field type. Since we know our data file is delimited by tabs (or whitespace) select the **Delimited** button. Also, be sure to specify to begin reading the data at line 3, not line 1 in the **Data Starts at Line** field. This is because there are two comment lines at the beginning of the file.

5. Click **Next**.

6. Now the **Define Delimiter/Fields** dialog box appears:



Number of Fields Per Line: 3

Delimiter Between Data Elements:

White Space   Comma   Colon
Semicolon   [✓ Tab]   Other:

Selected Records:

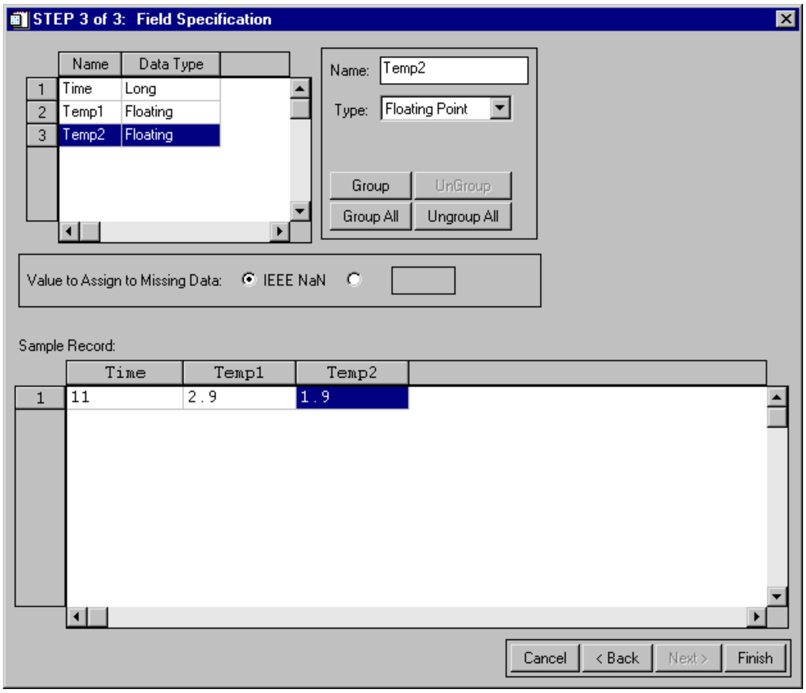
	Text
1	11 2.9 1.9
2	12 3.2 3.9
3	13 6.0 7.1
4	14 8.5 7.3
5	15 9.2 10.1
6	16 9.8 8.9
7	17 12.7 13.9
8	18 8.2 7.2
9	19 5.8 6.9

Figure 3-3: Selecting delimiter type in ASCII_TEMPLATE

7. At this dialog, be sure to select **Tab** as the delimiter between data elements since we know we have used tabs in the original file.

8. Now move on to the final dialog by clicking **Next**.

9. Now the **Field Specification** dialog box appears:



	Name	Data Type
1	Time	Long
2	Temp1	Floating
3	Temp2	Floating

Name: Temp2

Type: Floating Point

Value to Assign to Missing Data: [✓ IEEE NaN]

Sample Record:

	Time	Temp1	Temp2
1	11	2.9	1.9

Figure 3-4: Naming the data type sets in ASCII_TEMPLATE

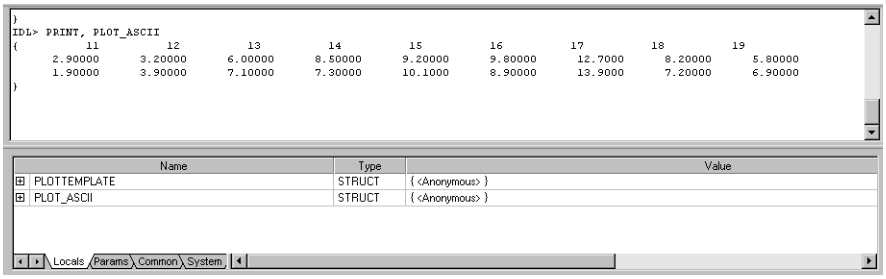
10. In this dialog, we will give a name to each data field for IDL to recognize each set. At the top of the box, click on the first row (row 1) and then name the data set by typing the name Time for the first set in the box.
11. Next, move on to the second row, naming this data set Temp1 for the first set of temperatures in the data set.
12. Finally, name the last data set Temp2. Click **Finish**.
13. Type the following line at the IDL Command Line to read in the file `plot.txt` using the template we've just created:

```
PLOT_ASCII=READ_ASCII(FILEPATH('plot.txt', SUBDIR = $
   ['examples', 'data']), TEMPLATE = PLOTTEMPLATE)
```

14. Then enter:

```
PRINT, PLOT_ASCII
```

You will see the following displayed:



```
}
IDL> PRINT, PLOT_ASCII
{          11          12          13          14          15          16          17          18          19
      2.90000     3.20000     6.00000     8.50000     9.20000     9.80000     12.7000     8.20000     5.80000
      1.90000     3.90000     7.10000     7.30000     10.1000     8.90000     13.9000     7.20000     6.90000
}
```

Name	Type	Value
⊞ PLOTTEMPLATE	STRUCT	{ <Anonymous> }
⊞ PLOT_ASCII	STRUCT	{ <Anonymous> }

Figure 3-5: PLOT_ASCII printed

Note

Note that PLOT_ASCII is a type STRUCT. When using READ_ASCII to read a file, the data is read into a structure variable. For more information on importing data into structure variables, see *Using IDL*.

Note

You may need to resize the IDL Output Log in order to see the PLOT_ASCII information displayed in correct columns.

Reading and Writing Binary Data

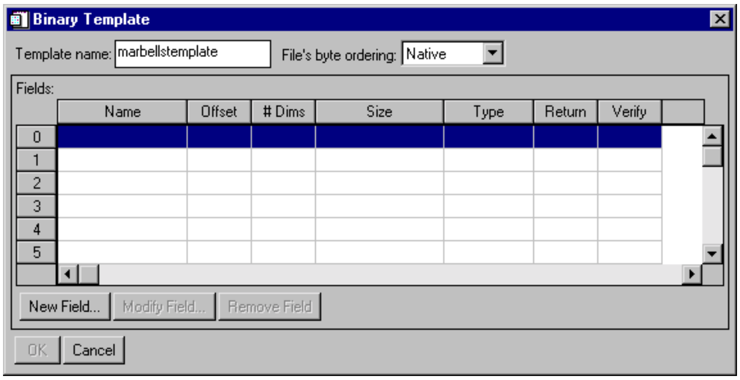
Reading data files into IDL is easy if you know the format in which the data is stored. Often, images are stored as arrays of bytes instead of a known format like JPEG or TIFF. These files we'll refer to as "Binary" files. The binary file that we will read in the following example contains an image of the Maroon Bells mountains, a group of mountains located in the Rocky Mountains of Colorado, stored as an integer array.

1. We will use the BINARY_TEMPLATE function in conjunction with the READ_BINARY function. At the IDL Command Line, enter the following:

```
MARBELLSTEMPLATE=BINARY_TEMPLATE(FILEPATH('surface.dat', $
   SUBDIR = ['examples', 'data']))
```

Note

Another way to import Binary data is to use the "Import Binary File" toolbar button on the IDLDE toolbar. To use this feature simply click the button and the dialog will appear so that you may select "surface.dat".



Template name: marbellstemplate

File's byte ordering: Native

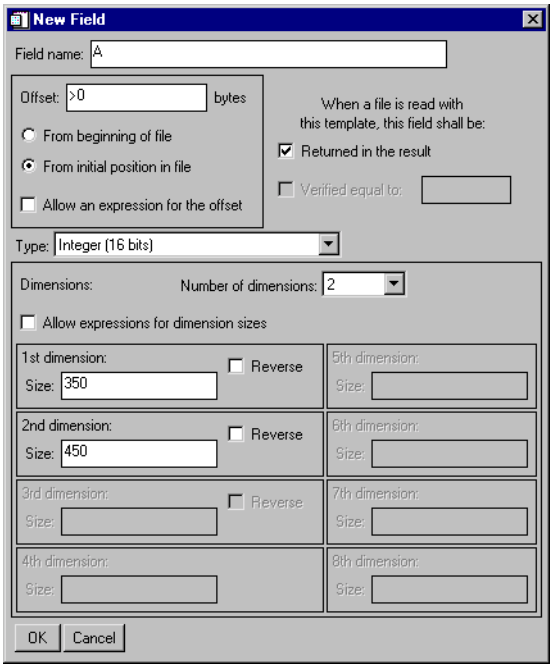
Figure 3-6: The binary template dialog

The binary template dialog box shown above appears.

2. In the **Template Name** field, enter "marbellstemplate" for the name of our new template.

3. In the **File's byte ordering** pull-down field, select "Little Endian" since we know that this file was created on an Intel processor-based machine. For more information about file byte ordering, see Chapter 15, "Reading and Writing Binary Data" in *Using IDL*.

4. Now we are ready to enter the field values, click the **New Field** button in the lower left corner of the dialog box.
5. When the **New Field** dialog appears, enter “A” as the field name. Verify the box in the upper right corner marked **Returned in the result** since we will want our data set returned at the time it is read.
6. At the **Number of Dimensions** pull-down menu, be sure to specify that we are dealing with a two-dimensional data set here. These data are contained in a 350 by 450 array, so we will enter these values in the two boxes marked **Size**.
7. Finally, let the binary template dialog know that we are dealing with Integer type data by specifying **Integer (16 bits)** at the **Type** pull-down menu. Click **OK**.


New Field

Field name: A

Offset: >0 bytes

From beginning of file [✓ From initial position in file]

Allow an expression for the offset

When a file is read with this template, this field shall be:

[✓ Returned in the result]

Verified equal to:

Type: Integer (16 bits)

Dimensions: Number of dimensions: 2

Allow expressions for dimension sizes

1st dimension: Size: 350 Reverse

2nd dimension: Size: 450 Reverse

3rd dimension: Size: Reverse

4th dimension: Size:

5th dimension: Size:

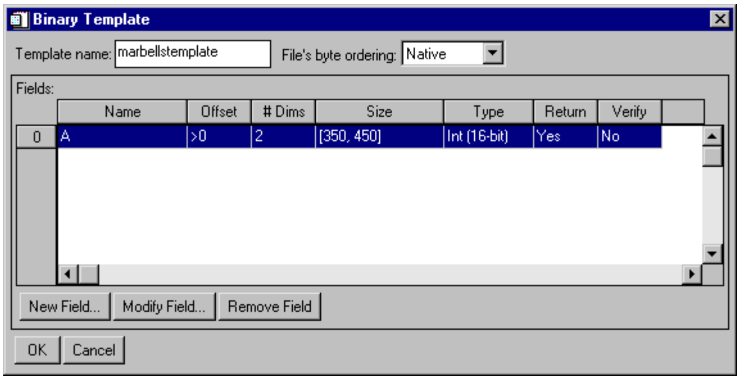
6th dimension: Size:

7th dimension: Size:

8th dimension: Size:

Figure 3-7: Modifying fields in binary template

Once you have entered the above data, the binary template dialog appears once again showing the specifications you have made. Your dialog should appear as the following figure.



Binary Template

Template name: marbellstemplate

File's byte ordering: Native

Fields:

	Name	Offset	# Dims	Size	Type	Return	Verify
0	A	>0	2	[350, 450]	Int (16-bit)	Yes	No

Figure 3-8: The "completed" binary template dialog

8. Now click **OK**.

Now we will use the READ_BINARY function to read the template we have just created.

9. At the IDL Command Line, enter:

```
MARBELLS_BINARY=READ_BINARY (FILEPATH('surface.dat', $
   SUBDIR=['examples', 'data']),TEMPLATE=MARBELLSTEMPLATE)
```

10. Now display the image by entering:

```
TVSCL, MARBELLS_BINARY.A
```

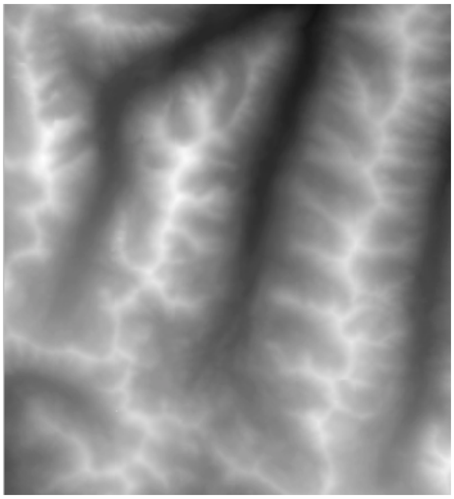


Figure 3-9: Surface.dat displayed using TVSCL

You can view an image in IDL with two different routines. The TV procedure writes an array to the display as an image without scaling. The TVSCL procedure displays the image with the color values scaled to use the whole color table.

Saving a Template

If you have multiple ASCII or Binary files of the same format, you can save your template so that you can reuse it. We'll demonstrate saving the template we created earlier in this section:

1. Save the ASCII template you have just created by entering:

   ```
   SAVE, PLOTTEMPLATE, FILENAME='MYPLOTTEMPLATE.dat'
   ```

2. Then, you can restore the template so that you can read another ASCII file:

   ```
   RESTORE, 'PLOTTEMPLATE.dat'
   ```

   This actually restores the variable named PLOTTEMPLATE which contains the template information.

   You can now read in another file using the READ_ASCII function by specifying `PLOTTEMPLATE.dat` for the TEMPLATE.

Note

You may also use an ASCII template to read another ASCII file provided that the data starts on the same line as the template specifies and that it is delimited in the same way as the template specifies.

Reading and Writing Images

Reading image files into IDL is also easy if you know the format in which the image is stored. First we must read in the image. Here we will use a TIFF format image of an aerial satellite view of Manhattan Island in New York City.

1. Enter the following at the IDL Command Line:

```
MYIMAGE=READ_TIFF(FILEPATH('image.tif',SUBDIR= $
   ['examples', 'data']))
```

   This command reads the image into memory.

2. Now display the image:

```
TV, MYIMAGE
```

Note

Another way to import image data is to use the “Import Image File” toolbar button on the IDLDE toolbar. To use this feature simply click the button and the dialog will appear so that you may select “image.tif”. However, this will name the image differently than shown in this example. For more information, see Chapter 13, “Reading and Writing Images” in *Using IDL*.

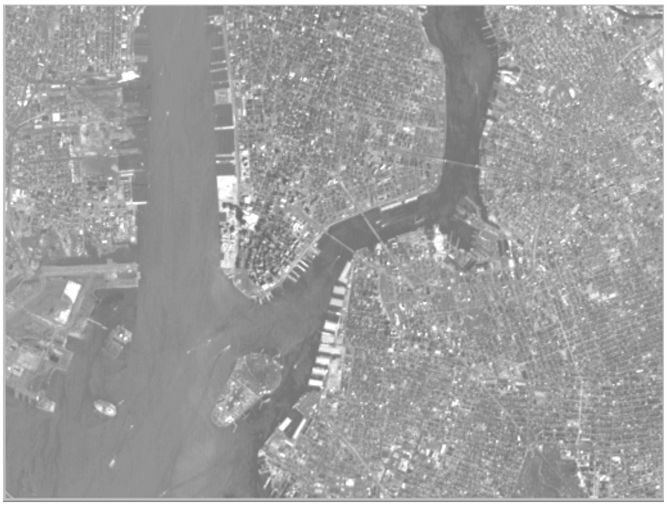


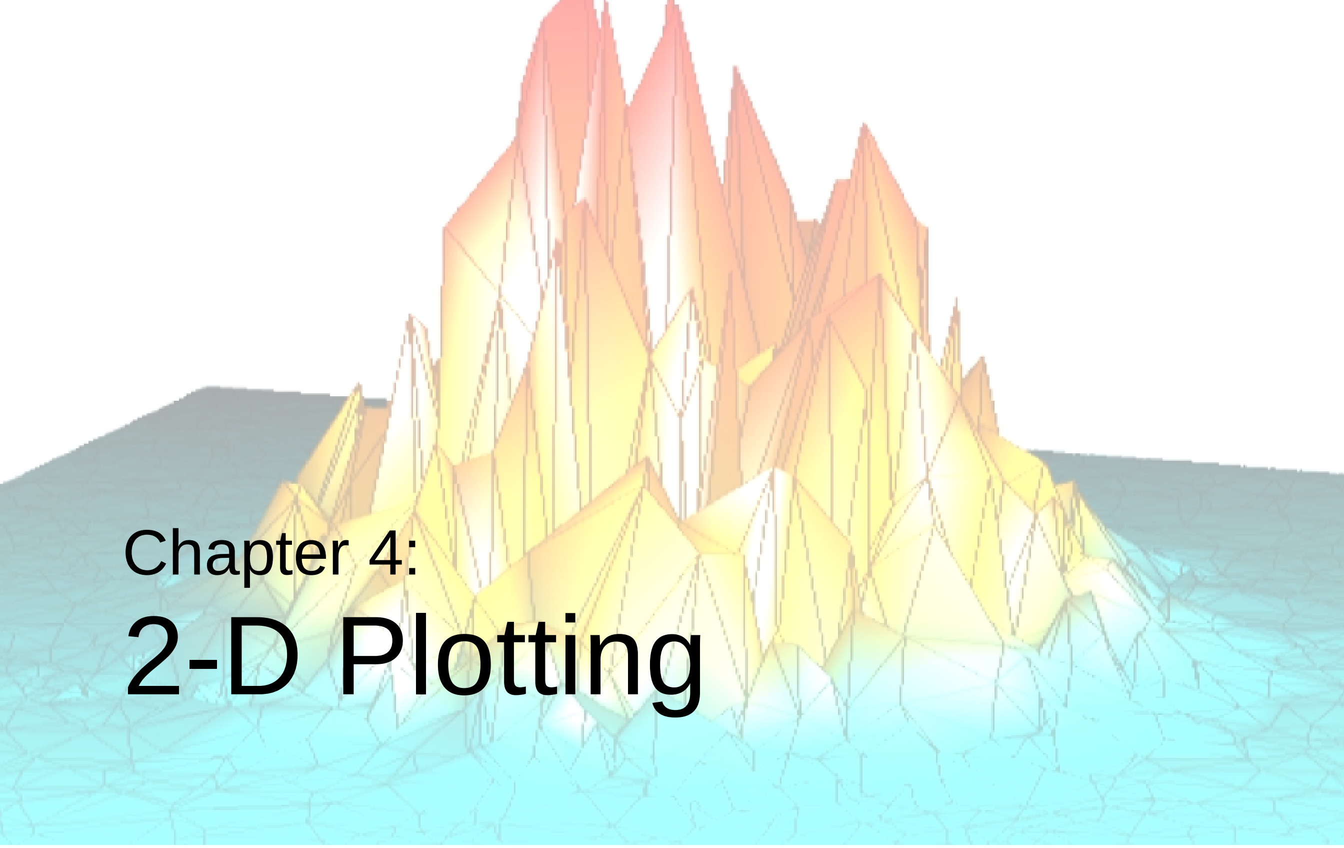
Figure 3-10: Reading and displaying an image file

3. Now, using IDL’s WRITE_TIFF command, rename and write the file:

   ```
   WRITE_TIFF, 'imagecopy.tif', MYIMAGE
   ```

More Information on IDL and Input/Output

For more information about IDL's input/output capabilities, see *Using IDL*. Also, for more detailed information on the functions and procedures you have seen in this chapter, see the *IDL Reference Guide*.



Chapter 4: 2-D Plotting

This chapter describes the following topics:

IDL and 2-D Plotting 40
Simple Plotting 41
Plotting with Data Sets 44
Other Plotting Capabilities 45
Using LIVE_PLOT 46
More Information on 2-D Plotting 48

IDL and 2-D Plotting

IDL makes plotting data easy. X versus Y plots can be displayed with a single command and multiple plots can be viewed at the same time. This tutorial demonstrates some of IDL's plotting capabilities. We will also examine how you enter statements at the IDL Command Line. This demonstrates IDL's interactive capability, and shows how easy it is to manipulate your data.

Note

To simplify obtaining useful results from the examples in this manual, create a bitmap buffer for your graphic windows and to use a maximum of 256 colors by entering the following command at the IDL command prompt:

```
DEVICE, RETAIN=2, DECOMPOSED=0
```

Simple Plotting

Simple plots can be charted using the PLOT procedure. Each call to PLOT establishes the plot window (the rectangular area enclosed by the axis), the plot region (the box enclosing the plot window and its annotation), the axis types (linear or logarithmic), and the scaling.

First, we'll plot a simple graph using a sine function. Use the FINDGEN function here to specify the dimensions of the array. The FINDGEN function returns a single-precision, floating-point array, with the specified dimension, where each element of the array is set to the value of its one-dimensional subscript.

1. First, create a value for the X axis:

```
X= 2*!PI/100 * FINDGEN(100)
```

2. Now, use PLOT to visualize the array:

```
PLOT, SIN(X)
```

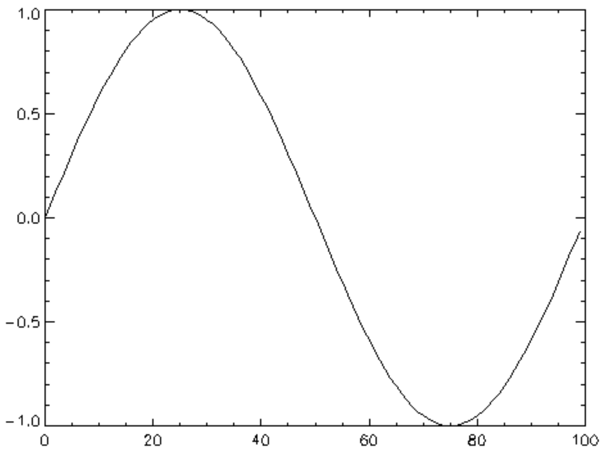


Figure 4-1: A simple sine wave using the PLOT command

Also, additional data can be added, as before, using the OPLOT procedure. Frequently, the color index, linestyle, or line thickness parameters are changed in each call to OPLOT to distinguish the data sets. The *IDL Reference Guide* contains a table describing the features you can change.

Using OPLOT

Now use OPLOT to plot the new information over the existing plot:

1. Plot at twice the frequency:

   ```
   OPLOT, SIN(2*X)
   ```

2. Plot at three times the frequency:

   ```
   OPLOT, SIN(3*X)
   ```

The results are shown in the following figure.

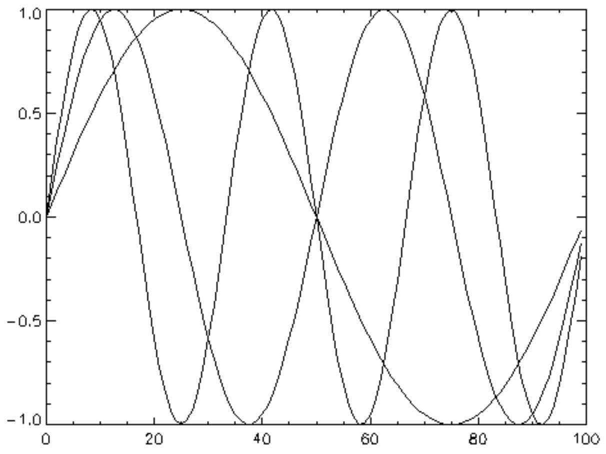


Figure 4-2: Graphing various data using the OPLOT command

Printing a Plot

IDL allows you to easily print the plot just created. Simply enter the following command lines shown.

1. First, save the original settings of your plotting environment:

   ```
   MYDEVICE=!D.NAME
   ```

2. Tell IDL that you wish to designate the printer to be the destination for the plot:

   ```
   SET_PLOT, 'printer'
   ```

3. Now plot again to the printer:

   ```
   PLOT, SIN(X)
   ```

4. Close the printing device:

   ```
   DEVICE, /CLOSE
   ```

5. Redesignate the original setting as the future destination for any plots:

   ```
   SET_PLOT, MYDEVICE
   ```

Note

If you are having problems printing on UNIX, be sure your printer is configured correctly. For more information on this see DIALOG_PRINTERSETUP in the *IDL Reference Guide*.

Plotting with Data Sets

To demonstrate IDL's capability to read a data set and plot it, we will use the template and data set used in the last chapter (Chapter 3, "Reading and Writing Data").

1. At the IDL Command Line, enter the following:

```
PLOT_ASCII=READ_ASCII(FILEPATH('plot.txt',SUBDIR= $
   ['examples', 'data']),TEMPLATE=PLOTTEMPLATE)
```

2. Plot the first set of data on temperatures which is stored in Temp1:

```
PLOT, PLOT_ASCII.TIME, PLOT_ASCII.TEMP1
```

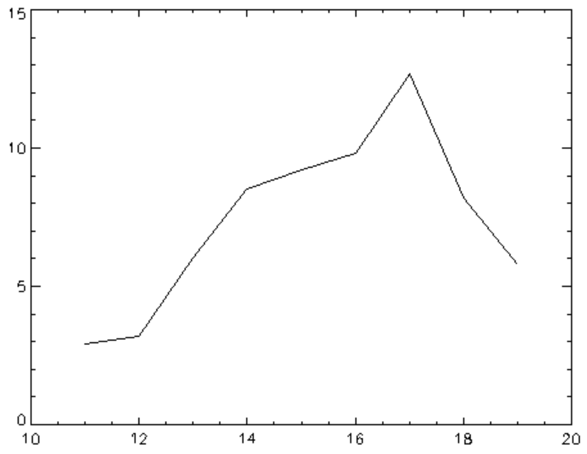


Figure 4-3: Plotting an existing data set using PLOT

Note

The file is read into a structure variable. For more information on importing data into structure variables, see *Using IDL*.

Other Plotting Capabilities

Now add titles to the simple plot graph using the TITLE, XTITLE, and YTITLE keywords. Using these simple keywords, IDL allows you to add a title to your plot as well as descriptive titles for your X and Y axis.

1. Plot with titles:

```
PLOT,PLOT_ASCII.TIME,PLOT_ASCII.TEMP1,TITLE= $
   'Temperature Over Time', XTITLE= $
   Time in Seconds',YTITLE='Temperature Celsius'
```

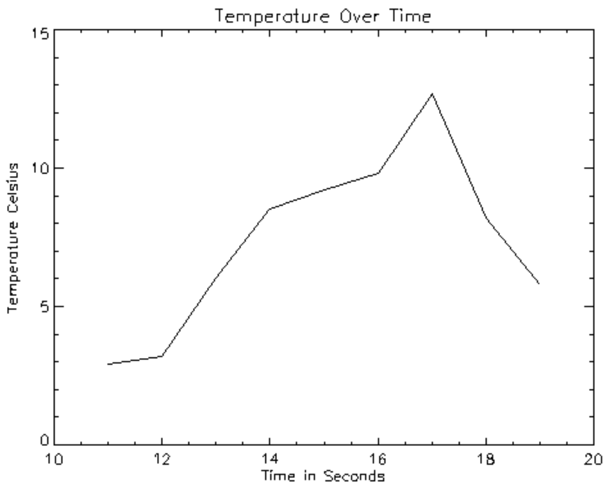


Figure 4-4: Adding titles to a plot using TITLE, XTITLE and YTITLE

Using LIVE_PLOT

The LIVE_PLOT procedure allows you to create an interactive plotting environment. Once plotted, you can double click on a section of the plot to display a properties dialog. A set of buttons in the upper left corner of the image window allows you to print, undo the last operation, redo the last "undone" operation, copy, draw a line, draw a rectangle, or add text. Using any of several auxiliary routines, you can control your LIVE window after it is created. See LIVE_PLOT in the *IDL Reference Guide* for an explanation.

1. Read in the same "Time over Temperature" data (See Chapter 3, "Reading and Writing Data" for instructions):

```
PLOT_ASCII=READ_ASCII(FILEPATH('plot.txt',SUBDIR= $
   ['examples', 'data']),TEMPLATE=PLOTTEMPLATE)
```

2. Create a LIVE plot

```
LIVE_PLOT,PLOT_ASCII.TEMP1,PLOT_ASCII.TEMP2, $
   NAME={data:['Temp1', 'Temp2']}
```

The result is shown in the following figure:

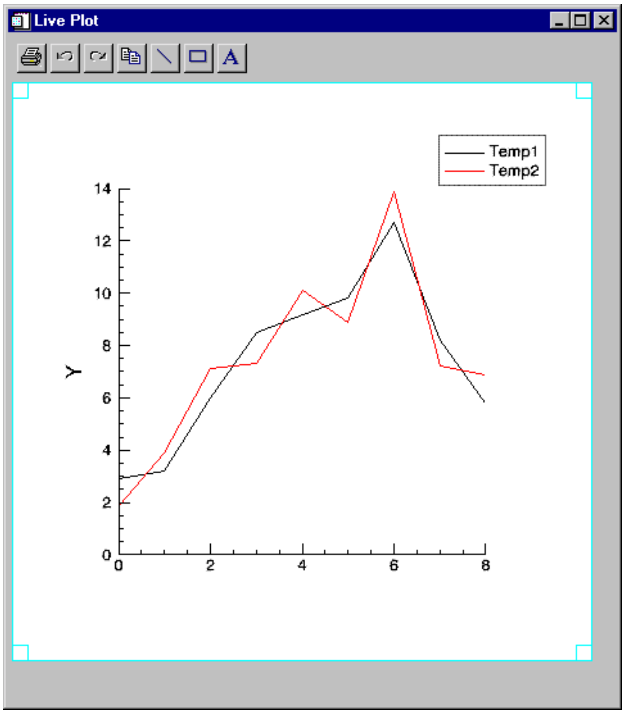
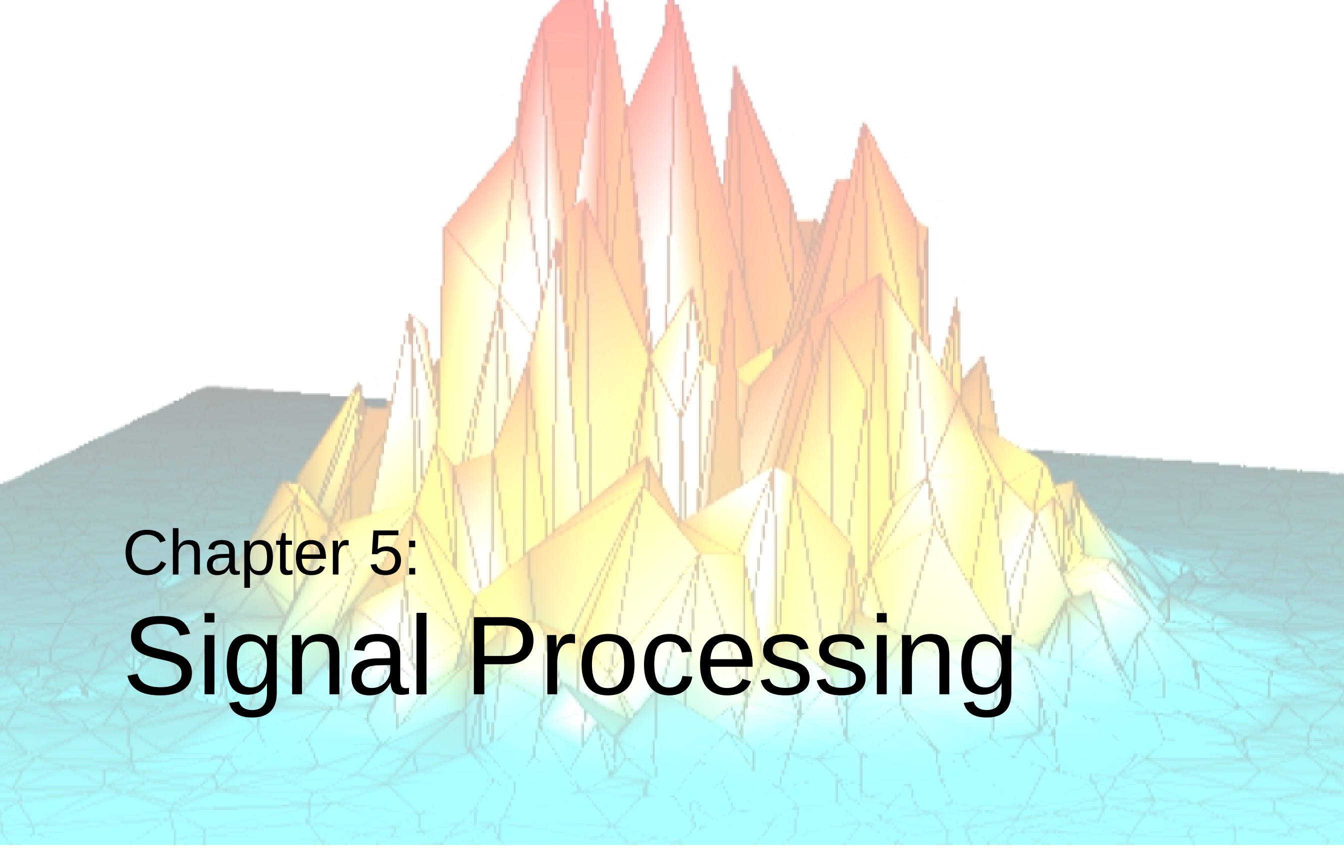


Figure 4-5: Using the LIVE_PLOT procedure

More Information on 2-D Plotting

IDL has many more plotting capabilities than the ones shown in this chapter. To take advantage of all of IDL's powerful capabilities in creating two-dimensional plots, look for more information in *Using IDL*.



Chapter 5: Signal Processing

This chapter describes the following topics:

IDL and Signal Processing 50
Creating a Data Set 51
Signal Processing with SMOOTH 54
Frequency Domain Filtering 55
Displaying the Results 58
More Information on Signal Processing . . . 60

IDL and Signal Processing

This chapter introduces you to IDL's digital signal processing tools. Most of the procedures and functions mentioned here work in two or more dimensions. For simplicity, only one-dimensional signals are used in the examples.

A signal, by definition, contains information. Any signal obtained from a physical process also contains noise. It is often difficult or impossible to make sense of the information contained in a digital signal by looking at it in its raw form—that is, as a sequence of real values at discrete points in time. Signal analysis transforms offer natural, meaningful, alternate representations of the information contained in a signal.

Creating a Data Set

First, we need to create a dataset to display.

1. Enter the following command to create a sinewave function with a frequency that increases over time and store it in a variable called `ORIGINAL`:

   ```
   ORIGINAL=SIN((FINDGEN(200)/35)^2.5)
   ```

   The FINDGEN function returns a floating-point array in which each element holds the value of its subscript, giving us the increasing “time” values upon which the sinewave is based. The sine function of each “time” value divided by 35 and raised to the 2.5 power is stored in an element of the variable `ORIGINAL`.

2. To view a quick plot of this dataset, shown in the following, enter:

   ```
   PLOT, ORIGINAL
   ```

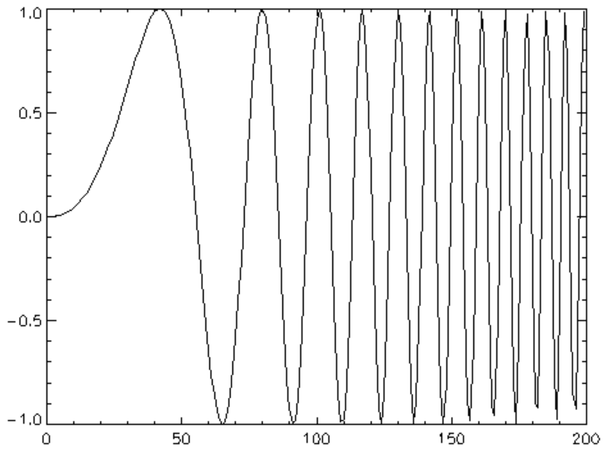


Figure 5-1: Plot of increasing frequency

3. Now add some uniformly-distributed random noise to this dataset and store it in a new variable:

```
NOISY=ORIGINAL+((RANDOMU(SEED,200)-.5)/ 2)
```

4. Now plot the array:

```
PLOT, NOISY
```

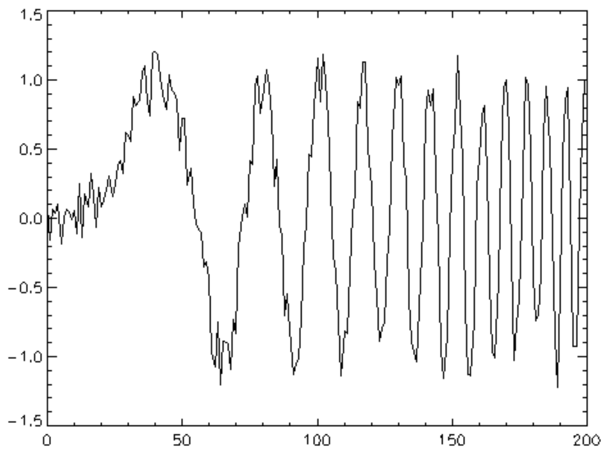


Figure 5-2: Plot of random noise

The RANDOMU function creates an array of uniformly distributed random values. The original dataset plus the noise is stored in a new variable called `NOISY`. This dataset looks more like real-world test data.

5. Display the original dataset and the noisy version simultaneously by entering the following commands:

```
PLOT, ORIGINAL, XTITLE="Time",YTITLE="Amplitude",THICK=3
```

6. Then overplot the previous data:

```
OPLOT, NOISY
```

The XTITLE and YTITLE keywords are used to create the X and Y axis titles. The OPLOT command plots the `NOISY` dataset over the existing plot of `ORIGINAL` without erasing. Setting the THICK keyword causes the default line thickness to be multiplied by the value assigned to THICK, so you can differentiate between the data. This result can be seen in the following figure.

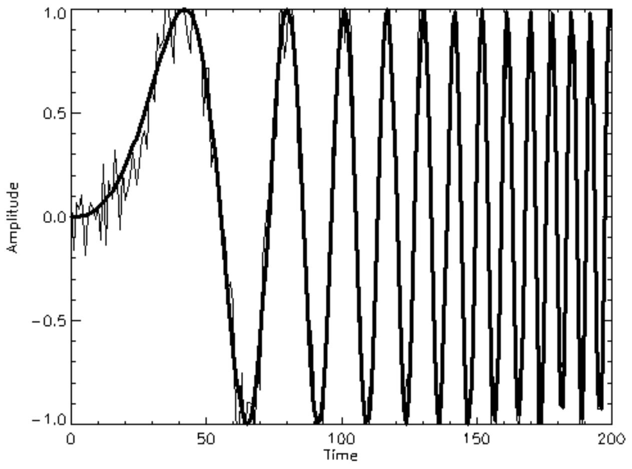


Figure 5-3: Combined plotting of datasets using the OPLOT command and THICK keyword

Signal Processing with SMOOTH

A simple way to smooth out the `NOISY` dataset is to use IDL's SMOOTH function. It returns an array smoothed with a boxcar average of a specified width.

1. Create a new variable to hold the smoothed dataset by entering the following command:

   ```
   SMOOTHED=SMOOTH(NOISY, 5)
   ```

2. Now plot your new data set:

   ```
   PLOT,SMOOTHED,TITLE='Smoothed Data'
   ```

   The TITLE keyword draws the title text centered over the plot. Notice that while SMOOTH did a fairly good job of removing noise spikes, the resulting amplitudes taper off as the frequency increases.

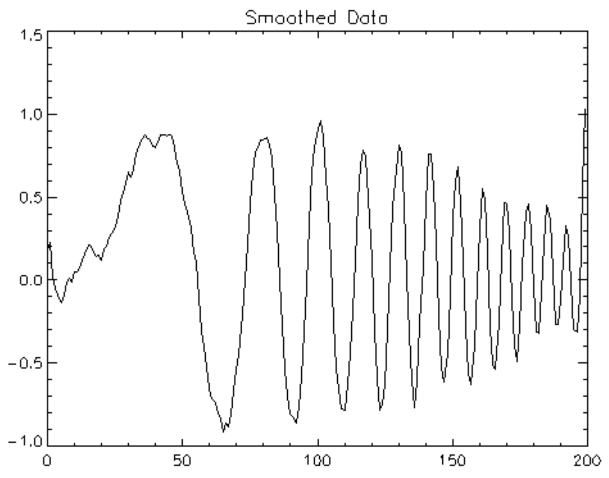


Figure 5-4: Using the SMOOTH command

Frequency Domain Filtering

Perhaps a better way to eliminate noise in the `NOISY` dataset is to use Fourier transform filtering techniques. Noise is basically unwanted high-frequency content in sampled data. Applying a lowpass filter to the noisy data allows low-frequency components to remain unchanged while high-frequencies are smoothed or attenuated. Construct a filter function by entering the following step-by-step commands:

1. Create a floating point array using FINDGEN which sets each element to the value of its subscript and stores it in the variable Y by entering:

   ```
   Y=[FINDGEN(100),FINDGEN(100)-100]
   ```

2. Next, make the last 99 elements of Y a mirror image of the first 99 elements:

   ```
   Y[101:199]=REVERSE(Y[0:98])
   ```

3. Now, create a variable filter to hold the filter function based on Y:

   ```
   filter=1.0/(1+(Y/40)^10)
   ```

4. Finally, plot:

   ```
   PLOT,FILTER
   ```

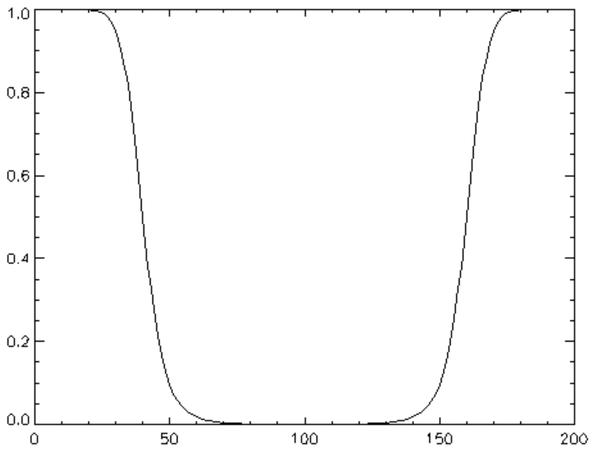


Figure 5-5: Constructing a filter function

To filter data in the frequency domain, we multiply the Fourier transform of the data by the frequency response of a filter and then apply an inverse Fourier transform to return the data to the spatial domain.

1. Now we can use a lowpass filter on the `NOISY` dataset and store the filtered data in the variable `lowpass` by entering:

```
LOWPASS=FFT(FFT(NOISY,1)*filter,-1)
```

2. Then plot:

```
PLOT, LOWPASS
```

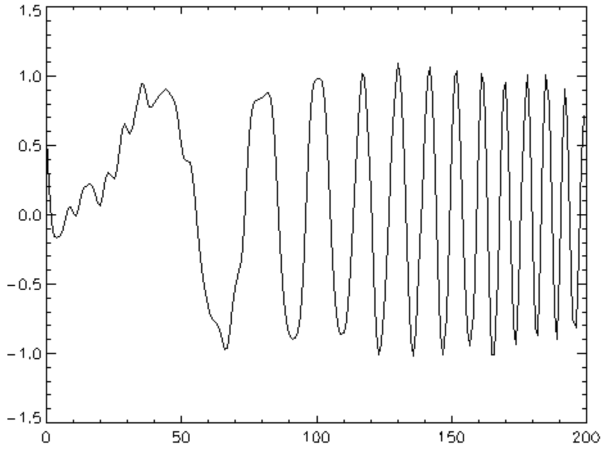


Figure 5-6: Using a LOWPASS filter

Note

Your plots may look slightly different due to the random number generator.

1. The same filter function can be used as a high-pass filter (allowing only the high frequency or noise components through) by entering:

```
HIGHPASS=FFT(FFT(NOISY,1)*(1.0-filter),-1)
```

2. Then plot:

```
PLOT, HIGHPASS
```

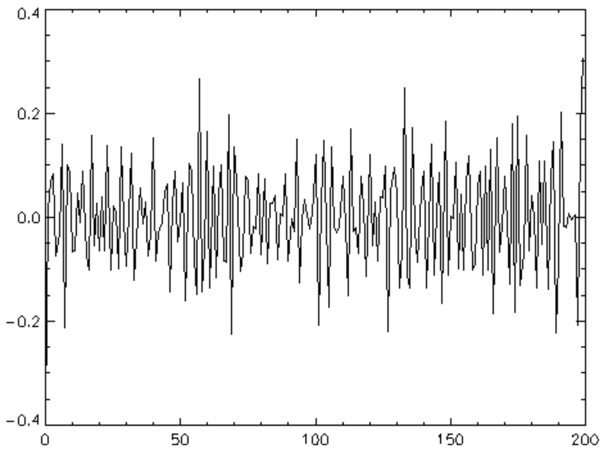


Figure 5-7: Using a highpass filter

Displaying the Results

Now look at all of the results at the same time. The plotting window can be split into six sections, making each section display a different plot. The system variable !P.MULTI tells IDL how many plots to put on a single page.

Enter the following lines to display a plotting window which shows all of the plots simultaneously.

1. To display all plots at the same time with two columns and three rows, enter:

   ```
   !P.MULTI=[0,2,3]
   ```

2. Now, display original dataset, upper-left:

   ```
   PLOT,ORIGINAL,TITLE='Original (Ideal) Data'
   ```

3. Next, display noisy dataset in the upper-right:

   ```
   PLOT,NOISY,TITLE='Noisy Data'
   ```

4. Display filter function, middle-left. The SHIFT function was used to show the filter's peak as centered.

   ```
   PLOT,SHIFT(filter,100),TITLE='Filter Function'
   ```

5. Now, display low-pass filtered dataset in the middle-right:

   ```
   PLOT,LOWPASS,TITLE='Lowpass Filtered'
   ```

6. Display high-frequency noise, lower-left:

   ```
   PLOT,HIGHPASS,TITLE='Highpass Filtered'
   ```

7. Finally, display the SMOOTH function dataset for comparison with the low-pass filtered data in the lower right.

```
PLOT, smoothed, TITLE = 'Smoothed with Boxcar average'
```

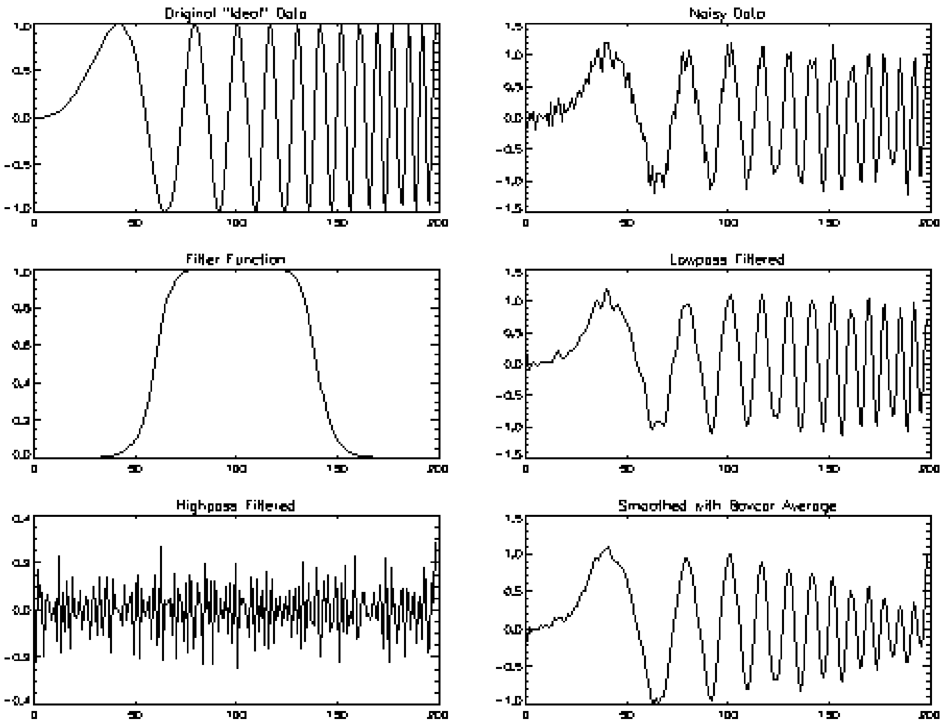


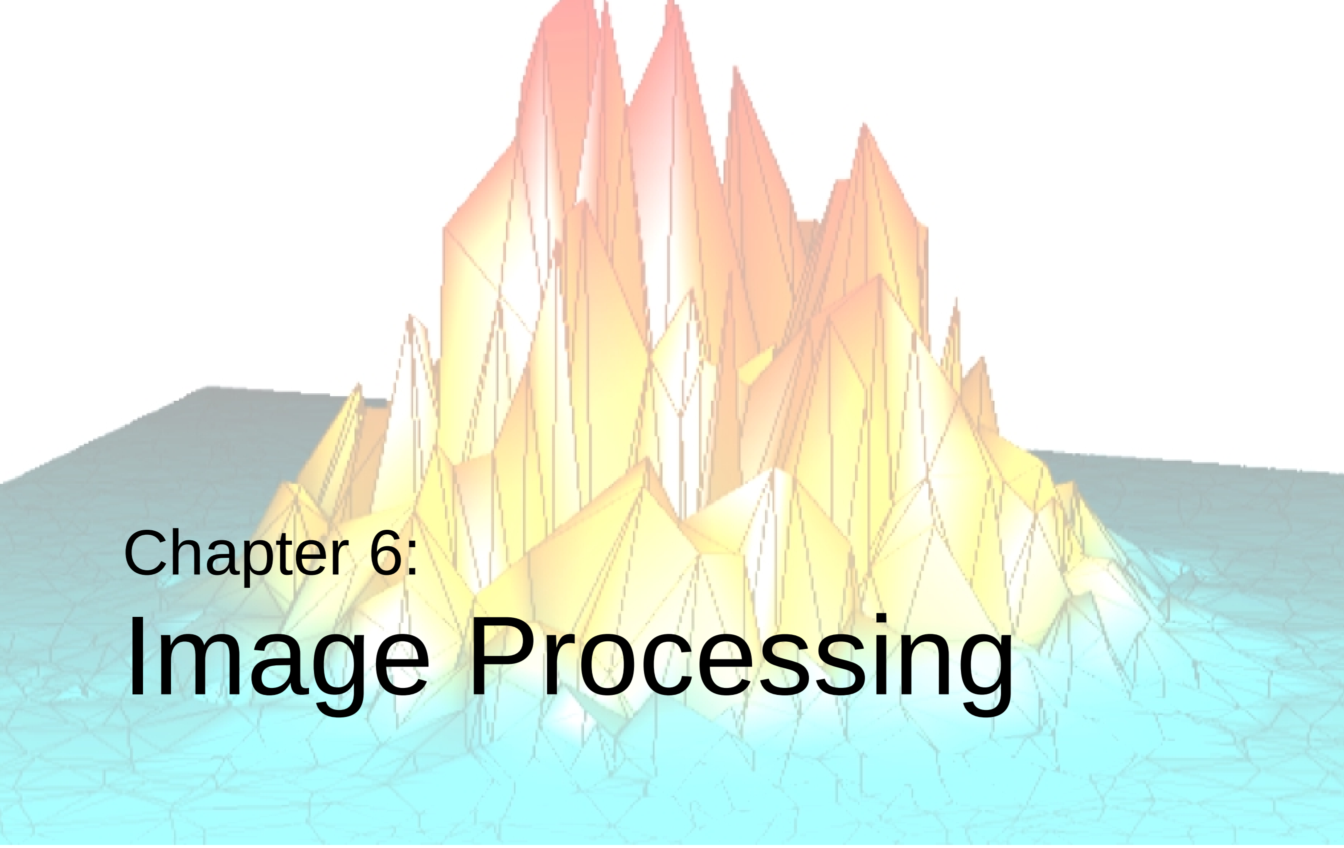
Figure 5-8: Display results using !P.MULTI to show six plots in one plotting window

8. Before continuing with the rest of the chapters, reset the plotting window to display a single image by entering the command:

```
!P.MULTI = 0
```

More Information on Signal Processing

Using just a few IDL commands, we have performed some complex and powerful signal processing tasks. IDL has many more signal processing abilities than the ones shown in this chapter. To take advantage of all of IDL’s powerful capabilities, look for more information in the *Using IDL* manual.



Chapter 6: Image Processing

This chapter describes the following topics:

IDL and Image Processing 62
Reading an Image 63
Displaying an Image 64
Resizing an Image 66
Resizing a Graphics Window 67
Contrast Enhancement 68
Smoothing and Sharpening 74
Other Image Manipulations 78
Extracting Profiles 80
Using LIVE_IMAGE 81
More Information on Image Processing . . . 82

IDL and Image Processing

IDL is an ideal tool for image processing because of its interactive operation, uniform notation, and array-oriented operators and functions. Images are easily represented as two-dimensional arrays in IDL and can be processed just like any other array. IDL also contains many procedures and functions specifically designed for image display and processing.

In this chapter, we will enter statements at the IDL Command Line. This demonstrates IDL's interactive capability, and shows how easy it is to manipulate your data.

Note

To simplify obtaining useful results from the examples in this manual, create a bitmap buffer for your graphic windows and to use a maximum of 256 colors by entering the following command at the IDL command prompt:

```
DEVICE, RETAIN=2, DECOMPOSED=0
```

Reading an Image

First we must import an image to be processed. Reading data files into IDL is easy. The file that we will read contains the image we used in Chapter 3, “Reading and Writing Data” of an aerial view above Manhattan in TIFF format.

1. Read the file by entering:

```
MYIMAGE=READ_TIFF(FILEPATH('image.tif',SUBDIR= $
   ['examples', 'data']))
```

Displaying an Image

You can view an image in IDL with two different routines. The TV procedure writes an array to the display as an image without scaling. Enter the commands below at the IDL Command Line.

Note

The default graphics window size is 640 by 512 pixels in size on a UNIX workstation and one-fourth of the display size on most Windows environments.

1. Display the image:

```
TV, MYIMAGE
```

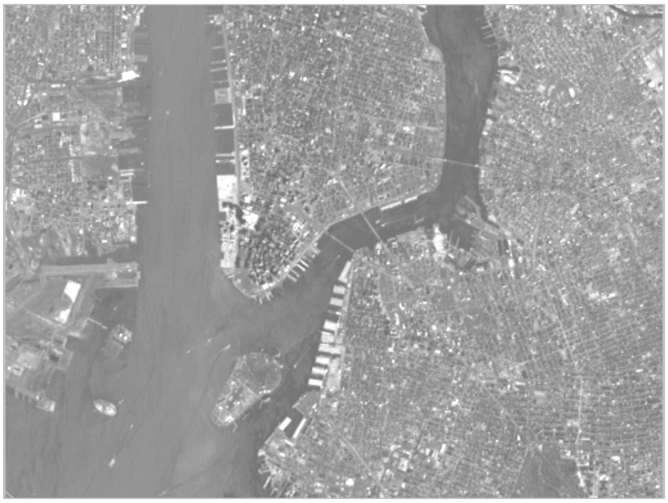


Figure 6-1: Displaying an image with TV

2. Enter WDELETE at the Command Line to dismiss the graphics window.

```
WDELETE
```

3. The TVSCL procedure displays the image with the color values scaled to use the whole color table. Display the scaled image:

```
TVSCL, MYIMAGE
```

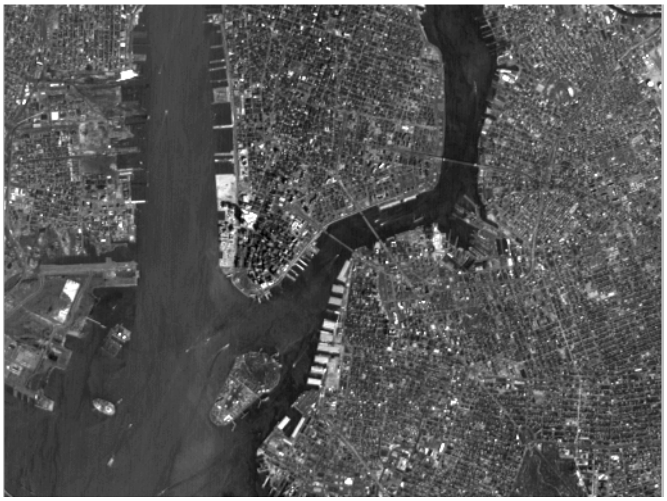


Figure 6-2: Displaying an image with TVSCL

4. Enter WDELETE at the Command Line to dismiss the graphics window.

```
WDELETE
```

Resizing an Image

The REBIN function in IDL makes it easy to resize a vector or array to new dimensions. The supplied dimensions must be proportionate (that is, integral multiples or factors) to the dimensions of the original image. Since our original image array here is 768 by 512, we'll need to decide the correct dimensions of our new resized image. If we want to resize the image to half the original size then simply take half of the array's original dimensions.

1. Create a new image with new dimensions using the REBIN function:

```
NEWIMAGE=REBIN(MYIMAGE,384,256)
```

2. Now display the image:

```
TV, NEWIMAGE
```

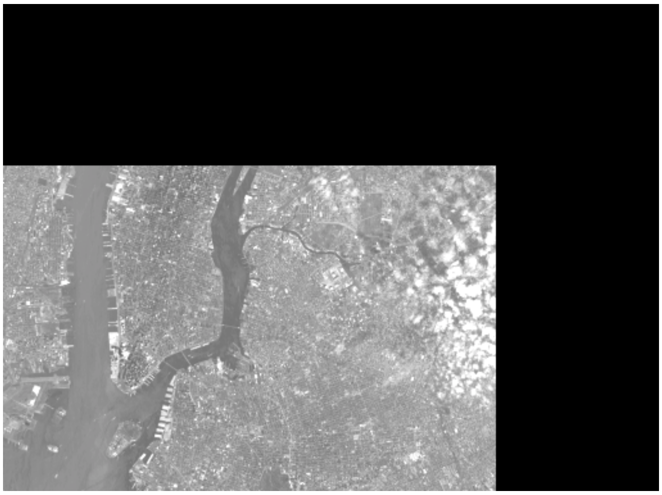


Figure 6-3: MYIMAGE resized to one half the original array size

3. Enter WDELETE at the Command Line to dismiss the graphics window.

```
WDELETE
```

Note

The CONGRID function also shrinks or expands the size of an array. CONGRID differs from REBIN in that where REBIN requires that the new array size must be an integer multiple of the original size, CONGRID will resize an array to any arbitrary size. For more information, see CONGRID in the *IDL Reference Guide*.

Resizing a Graphics Window

IDL automatically creates a window for displayed graphics if one does not already exist. You can use the WINDOW command to create new windows with custom sizes.

1. To display Manhattan in a larger graphics window, enter:

   ```
   WINDOW,0,XSIZE=800,YSIZE=600
   ```

2. Then enter:

   ```
   TV,MYIMAGE
   ```

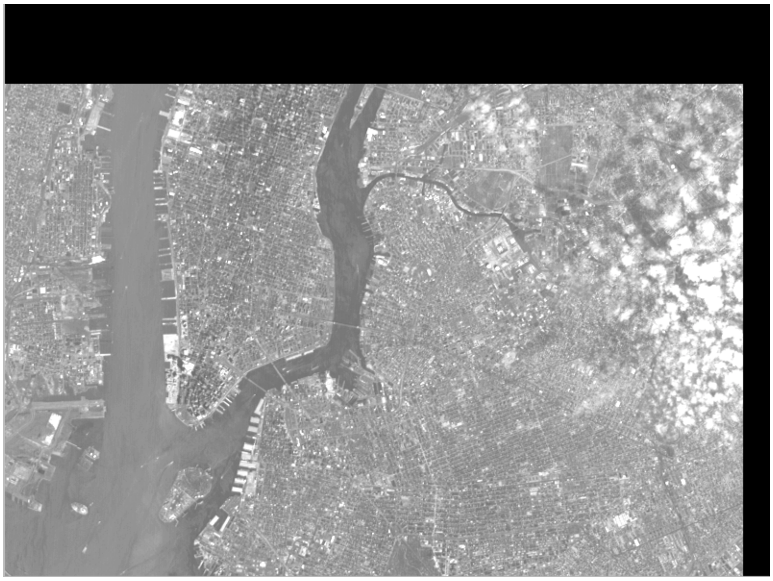


Figure 6-4: Visualizing a graphic through a larger graphic window

3. Enter WDELETE at the Command Line to dismiss the graphics window for the next graphic.

   ```
   WDELETE
   ```

The WINDOW command above creates a new version of window number 0 that is 800 pixels wide (specified with the XSIZE keyword) and 500 pixels tall (specified with the YSIZE keyword).

Contrast Enhancement

In order to improve the look of an image, sometimes all that is necessary is a change in how the colors are represented. IDL provides several ways to manipulate the contrast.

Thresholding

One of the simplest contrast enhancements that can be performed on an image is thresholding. Thresholding produces a two-level mapping from all of the possible intensities into black and white. The IDL relational operators, EQ, NE, GE, GT, LE, and LT, return a value of 1 if the relation is true and 0 if the relation is false. When applied to images, the relation is evaluated for each pixel and an image of 1’s and 0’s results.

1. To display the pixels in the image that have values greater than 140 as white and all others as black, as shown in the following, enter:

```
TVSCL,MYIMAGE GT 140
```

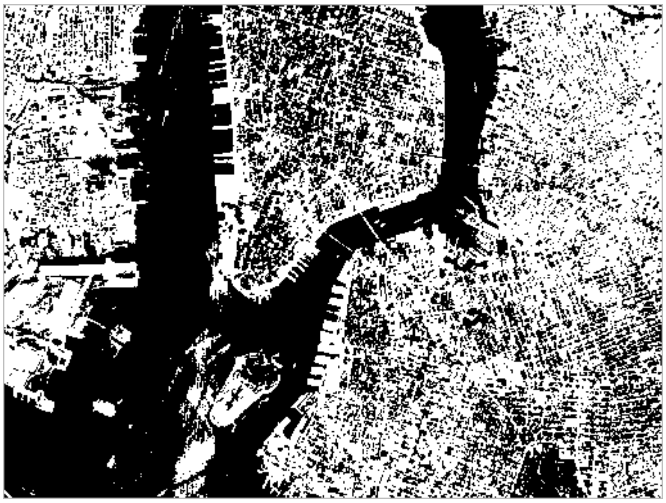


Figure 6-5: Image with all values greater than 140 shown as white

2. Similarly, the pixels that have values less than 140 can be displayed as white, as shown, by entering the command:

   ```
   TVSCL,MYIMAGE LT 140
   ```

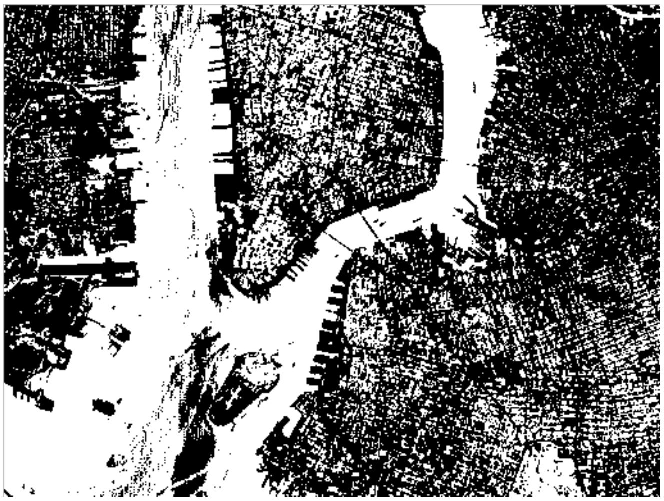


Figure 6-6: Image with all values less than 140 shown as white

In many images, the pixels have values that are only a small subrange of the possible values. By spreading the distribution so that each range of pixel values contains an approximately equal number of members, the information content of the display is maximized, as shown in the following.

3. The HIST_EQUAL function performs this redistribution on an array. To display a histogram-equalized version of myimage, enter the following:

   ```
   TV, HIST_EQUAL(myimage)
   ```

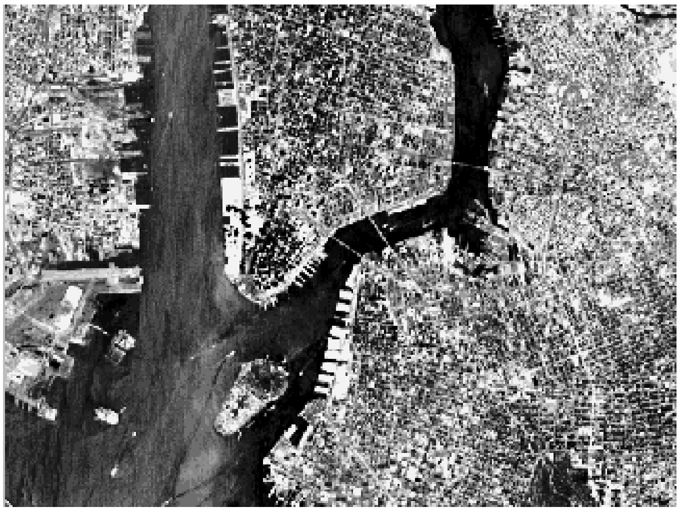


Figure 6-7: A histogram-equalized version of the image

Scaling Pixel Values

Another way to enhance the contrast of an image is to scale a subrange of pixel values to fill the entire range of displayed brightnesses. The > operator, the IDL maximum operator, returns a result equal to the larger of its two parameters. The following commands contrast the maximum and minimum operators.

1. Scale pixels with a value of 100 or greater into the full range of displayed brightnesses:

```
TVSCL,MYIMAGE > 100
```

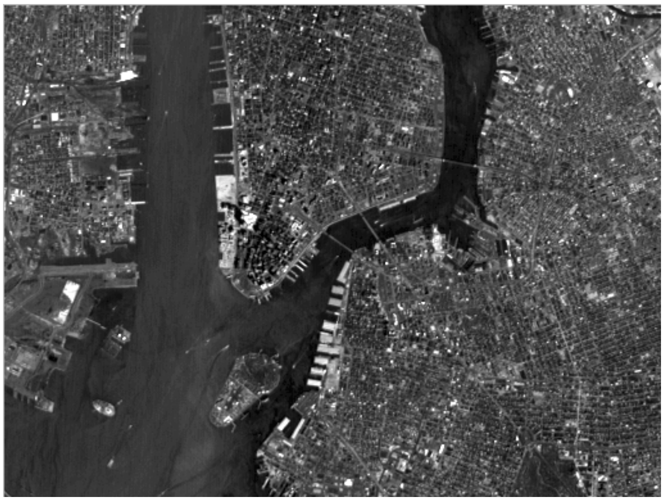


Figure 6-8: Image with pixels >100 scaled to full range of brightness

2. Scale pixels with a value less than 140 into the full range of brightnesses.

```
TVSCL,MYIMAGE < 140
```

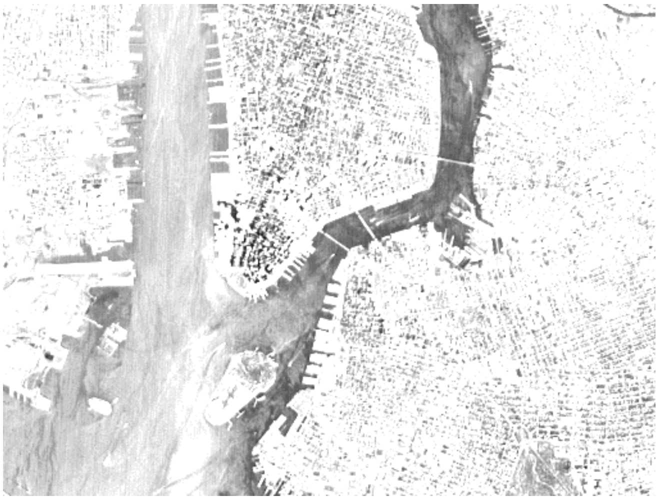


Figure 6-9: Image with pixels <140 scaled to full range of brightness

3. The minimum and maximum operators can be used together for more complicated contrast enhancements. Set the minimum brightness to 140, set the maximum brightness to 200, scale myimage and display it by entering:

```
TVSCL,MYIMAGE >140<200
```

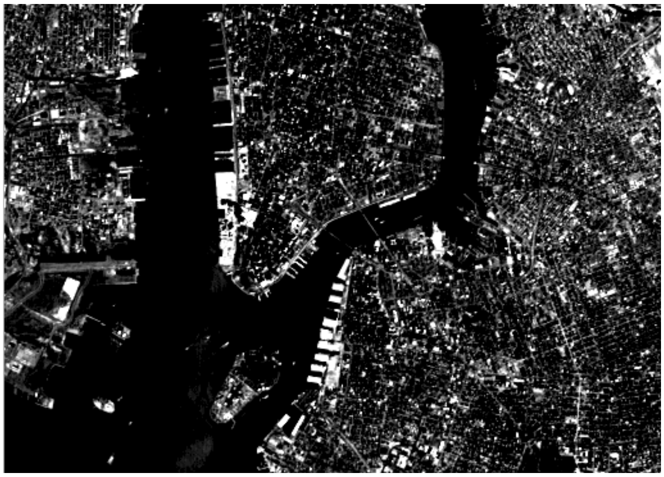


Figure 6-10: Image with minimum brightness at 140 and maximum brightness at 200

Note

Although this command illustrates the use of the IDL minimum and maximum operators, the same function can be executed more efficiently by IDL with the command:

```
TV, BYTSCL(MYIMAGE,MIN=140,MAX=200,TOP=!D.TABLE_SIZE)
```

Smoothing and Sharpening

Images can be rapidly smoothed to soften edges or compensate for random noise in an image using IDL's SMOOTH function. SMOOTH performs an equally weighted smoothing using a square neighborhood of an arbitrary odd width, as shown below.

1. Display myimage smoothed using a 7 by 7 area:

```
TVSCL,SMOOTH(MYIMAGE,7)
```

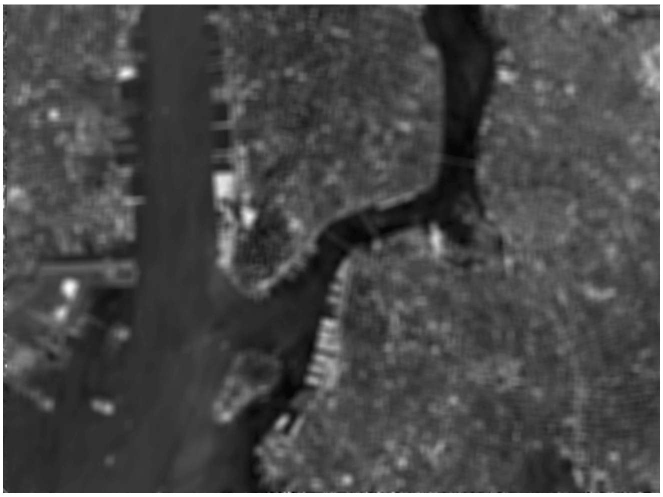


Figure 6-11: Smoothing with SMOOTH

Unsharp Masking

The previous image looks a bit blurry because it contains only the low frequency components of the original image. Often, an image needs to be sharpened so that edges or high spatial frequency components of the image are enhanced. One way to sharpen an image is to subtract a smoothed image containing only low-frequency components from the original image. This technique is called *unsharp masking*.

1. Unsharp mask and display image:

```
TVSCL, FLOAT(MYIMAGE)-SMOOTH(MYIMAGE,7)
```

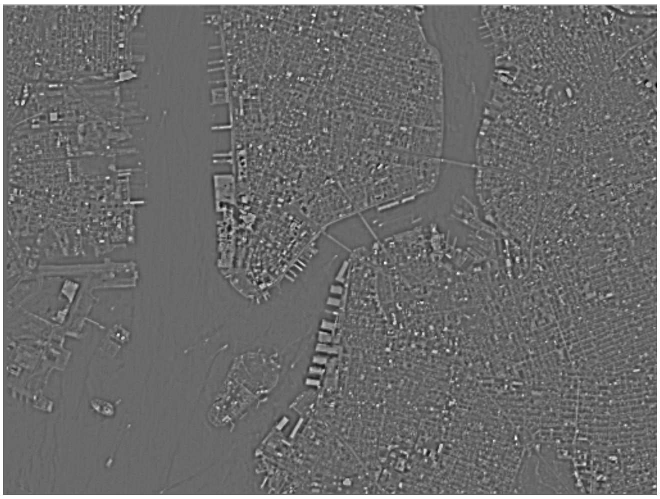


Figure 6-12: Unsharp masking

This command subtracts a smoothed version of the image from the original, scales the result, and displays it, as shown previously.

Sharpening Images with Differentiation

IDL has other built-in sharpening functions that use differentiation to sharpen images. The ROBERTS function returns the Roberts gradient of an image. Enter the following commands:

1. Create a new variable, R, that contains the Roberts gradient of myimage:

```
R=ROBERTS(MYIMAGE)
```

2. Display array R:

```
TVSCL, R
```

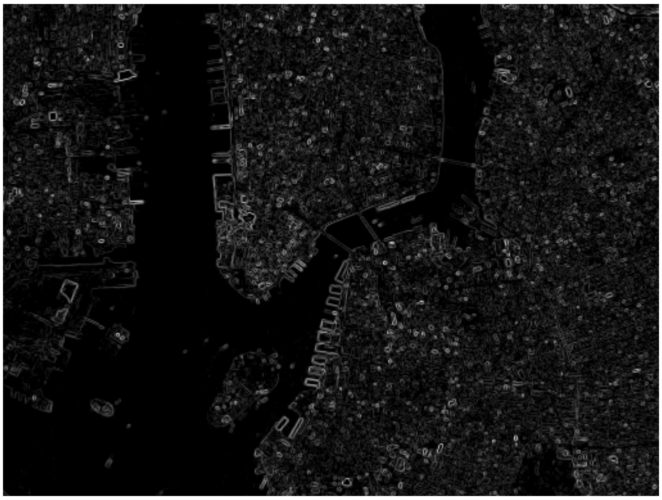


Figure 6-13: Roberts gradient of myimage

Another commonly used gradient operator is the Sobel operator. IDL's SOBEL function operates over a 3 by 3 region, making it less sensitive to noise than some other methods. Enter the following commands.

1. Create a Sobel sharpened version of the image:

```
SO=SOBEL(MYIMAGE)
```

2. Display the sharper image:

```
TVSCL, SO
```

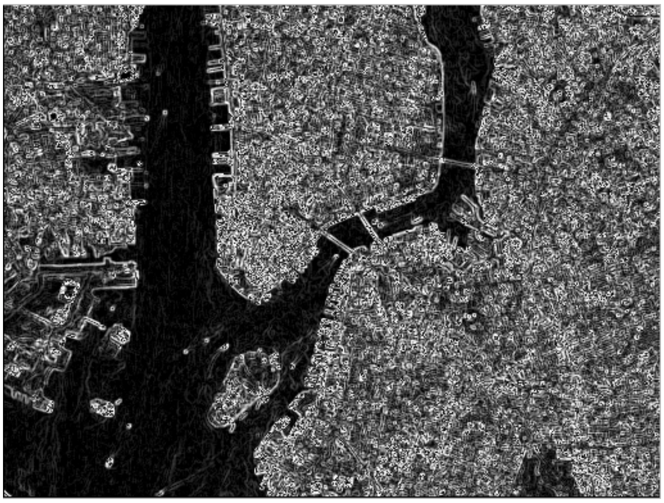


Figure 6-14: Sobel sharpened version of myimage

Loading Different Color Tables

Try loading some of the pre-defined IDL color tables to make this image more visible. While the graphics window is visible, type XLOADCT at the IDL Command Input Line. The XLOADCT widget application appears. Select a color table from the field; the window will reflect the color scheme. Click “Done” to accept a color table. When you are finished looking at the effects of different tables, click on the first color table in the field, B-W Linear, and click “Done” to load the original black and white color table.

Note

If you load a new color table while an image is still being displayed on a 24-bit (true) color display, you will need to close the image and reload it in IDL in order to see the new image displayed in the new color scheme. In an 8-bit (pseudo) color display however, you will not need to re-display the image as the color change will be immediate.

Other Image Manipulations

Sections of images can be easily displayed by using subarrays.

1. Erase the current display, create a new array that contains Manhattan and display it by entering:

```
ERASE
E=MYIMAGE[100:300, 150:250]
```

2. Then enter:

```
TV, E
```

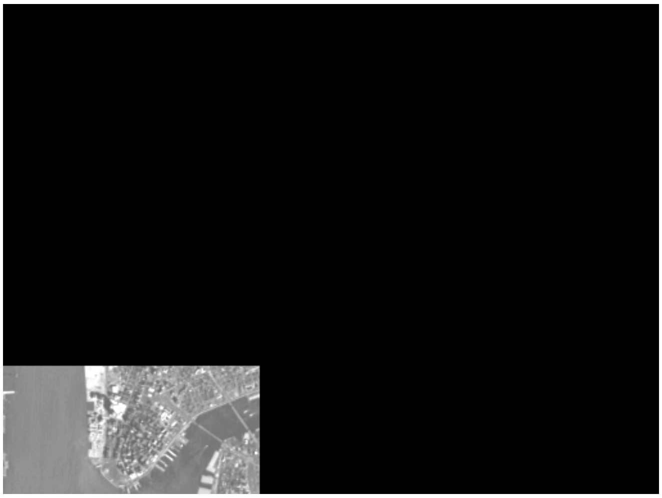


Figure 6-15: Displaying a section of an image

3. Enter WDELETE at the Command Line to dismiss the graphics window.

```
WDELETE
```

Rotating an Image

Simple rotation in multiples of 90 degrees can be accomplished with the ROTATE function.

1. Rotate the image by 90 degrees, as shown below, by entering:

   ```
   R=ROTATE(E,1)
   ```

2. Now enter to display:

   ```
   TVSCL, R
   ```

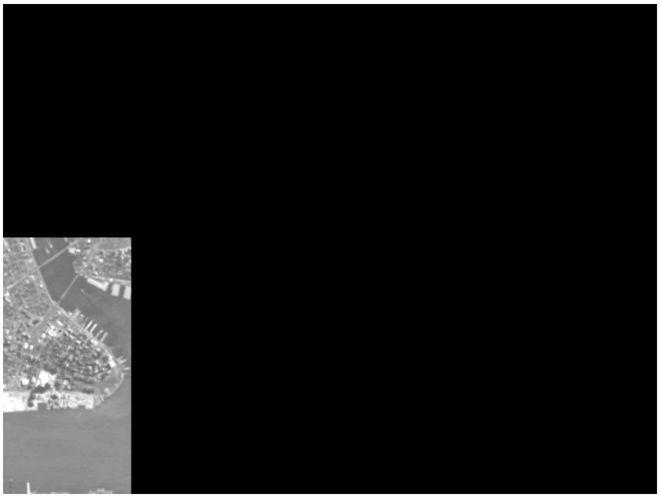


Figure 6-16: The image rotated by 90 degrees

The second parameter of ROTATE is an integer from 1 to 8 that specifies which one of the eight possible combinations of rotation and axis reversal to use.

3. Enter WDELETE at the Command Line to dismiss the graphics window for the next graphic.

   ```
   WDELETE
   ```

Extracting Profiles

Another useful image processing tool is the PROFILES routine. This routine interactively draws row or column profiles of an image. It allows you to view an image and an X-Y plot of the pixel brightnesses in any row or column of the image simultaneously.

1. Use the PROFILES routine with the rotated image that you just displayed by entering the following:

```
PROFILES, R
```

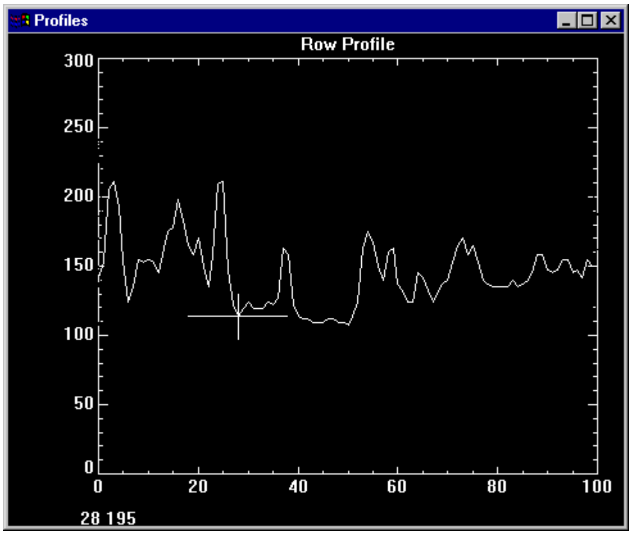


Figure 6-17: Viewing an image and an X-Y plot of the pixel brightnesses in any row or column

A new window for displaying the profiles appears. Move the cursor in the window containing the image R to display the profiles of different rows and columns.

2. Click the left mouse button while the cursor is in the image window to switch between displaying row and column profiles.
3. Click the right mouse button while the cursor is in the image window to exit the PROFILES routine.

Using LIVE_IMAGE

The LIVE_IMAGE procedure displays visualizations and allows interactive manipulation using the mouse and keyboard.

1. To display the same image of New York City that we used in Chapter 3, "Reading and Writing Data", this time using LIVE_IMAGE, enter:

```
LIVE_IMAGE, MYIMAGE
```

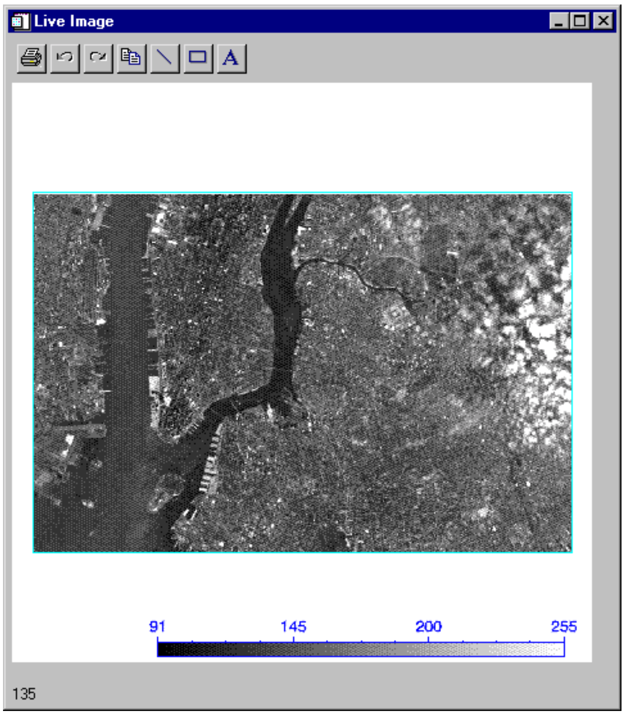


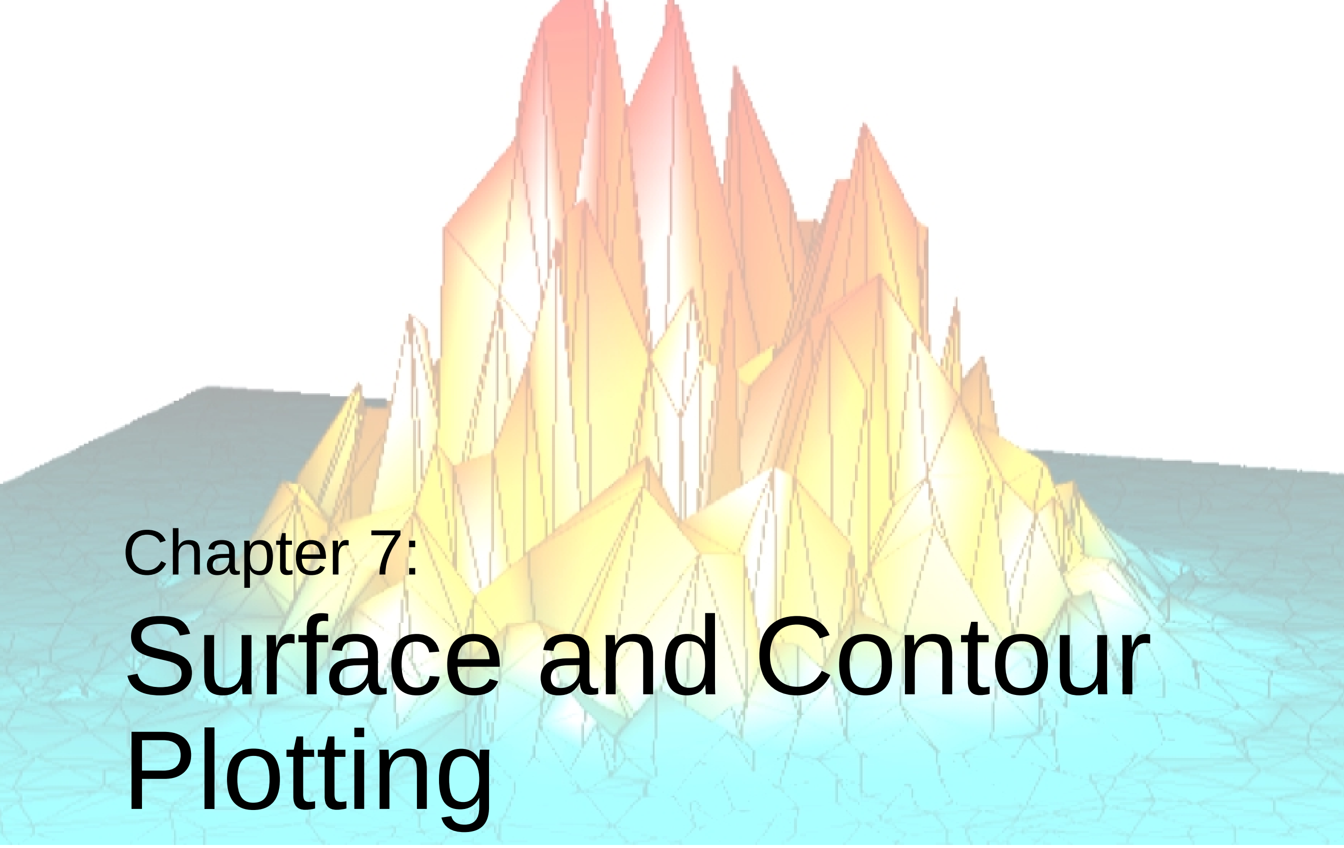
Figure 6-18: MYIMAGE shown using LIVE_IMAGE

You can click once on the image and then drag the cursor across the image to read the image values.

You may also double-click on the image to display a properties dialog. The set of buttons in the upper left corner of the image window allows you to print, undo the last operation, redo the last "undone" operation, copy, draw a line, draw a rectangle, or add text. The LIVE_IMAGE window may also be resized using the mouse.

More Information on Image Processing

IDL offers much more in the area of Image Processing. To learn more including some of the new functionality added to IDL on image display and image processing, see *Using IDL*.



Chapter 7: Surface and Contour Plotting

This chapter describes the following topics:

IDL and Surface and Contour Plotting 84
Reading a Dataset to Plot 85
Displaying a Surface 86
Displaying a Shaded Surface 88
Displaying a Contour 91
Plotting with SHOW3 96
Using LIVE_SURFACE for Plotting 97
More Information on 3-D Plotting 98

IDL and Surface and Contour Plotting

IDL provides many techniques for visualizing two-dimensional arrays, including contour plots, wire-mesh surfaces, and shaded surfaces. This chapter demonstrates just a few of the commands for visualizing data in three dimensions.

Note

To simplify obtaining useful results from the examples in this manual, create a bitmap buffer for your graphic windows and to use a maximum of 256 colors by entering the following command at the IDL command prompt:

```
DEVICE, RETAIN=2, DECOMPOSED=0
```

Reading a Dataset to Plot

First, we need to create a two-dimensional dataset to visualize. For these examples we will use the binary dataset of the Maroon Bells mountains that we used in the section "Reading and Writing Binary Data" on page 31. You will need to turn back to that section to once again use the BINARY_TEMPLATE and READ_BINARY functions to read this data set for use. Once you have read the file into IDL, you are ready to move on to visualizing the data set three-dimensionally.

Displaying a Surface

First, view the array MARBELLS_BINARY.A as a three-dimensional, “wire-mesh” surface. Use the CONGRID procedure initially to resample the data set so that the “mesh” can be displayed at a size visible to the human eye.

1. Here resample the array size to 35 by 45, or one tenth its original size. To do this enter:

```
A=CONGRID(MARBELLS_BINARY.A,35,45)
```

2. Now we are ready to visualize the mesh using the SURFACE command:

```
SURFACE, A
```

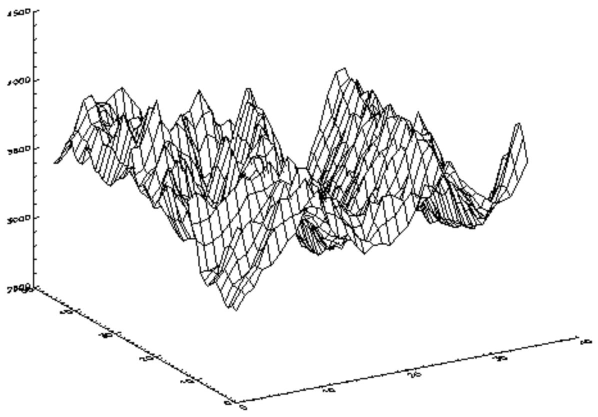


Figure 7-1: Surface plot with default angles

The SURFACE command can be used to view your data from different angles. AX and AZ are plotting keywords that are used to control the SURFACE command. The keyword AX specifies the angle of rotation of the surface (in degrees towards the viewer) about the X axis. The AZ keyword specifies the rotation of the surface in degrees counterclockwise around the Z axis.

3. View the array from a different angle by entering the following command:

```
SURFACE, A, AX = 70, AZ = 25
```

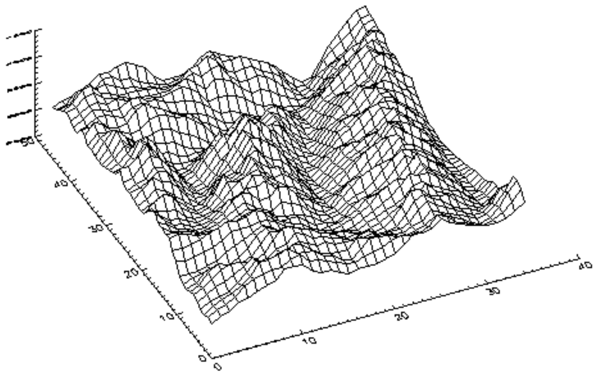


Figure 7-2: Surface plot showing different angles

Displaying a Shaded Surface

You can also view a two-dimensional array as a light-source shaded surface.

1. First, load one of the pre-defined IDL color tables by entering:

```
LOADCT, 3
```

2. To view the light-source shaded surface, shown in the following, simply enter the command:

```
SHADE_SURF, A
```

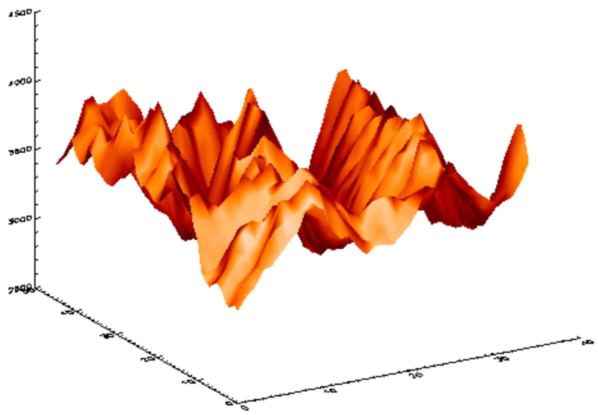


Figure 7-3: Surface plot with light-source shaded

3. To look at the array from another angle, enlarge the label text, and add a title. Again, keywords are used to control certain features of the shaded surface plot. The AX and AZ keywords control the viewing angle, just as they did with the SURFACE command.

   The CHARSIZE keyword controls the size of plotted text. The TITLE keyword was used to add the title “Shaded Surface Representation”.

```
SHADE_SURF,A,AX=45,AZ=20,CHARSIZE=1.5, $
   TITLE='Shaded Surface Representation'
```

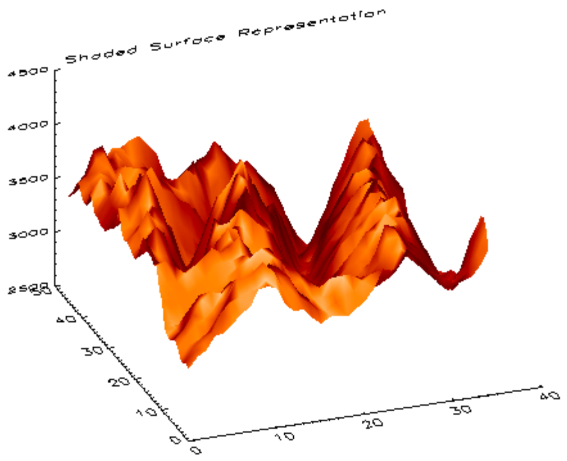


Figure 7-4: Surface plot with annotated surface plot

4. You can create a different kind of shaded surface, where the shading information is provided by the elevation of each point. Now different shading colors on the plot correspond to different elevations (the BYTSCL function scales the data values into the range of bytes).

   You could also specify a different array for the shading colors.

   ```
   SHADE_SURF,A,SHADE=BYTSCL(A)
   ```

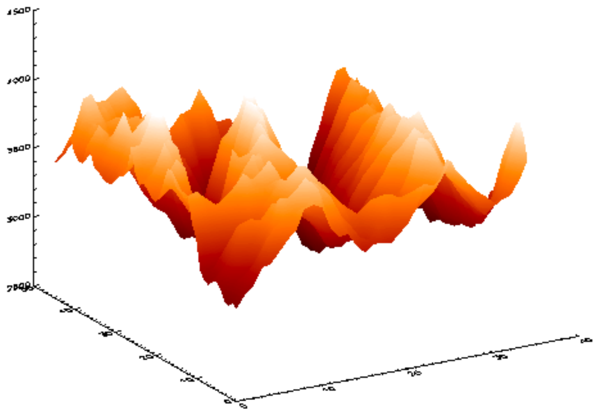


Figure 7-5: Byte-scaled surface plot

5. You can plot a wire-frame surface of the Maroon Bells (mountains) right over the existing plot. The XSTYLE, YSTYLE, and ZSTYLE keywords are used to select different styles of axis. Here, SURFACE is set to not draw the X, Y, and Z axis because they were already drawn by the SHADE_SURF command.

   The /NOERASE keyword allows the SURFACE plot to be drawn over the existing SHADE_SURF plot. Enter the following:

   ```
   SURFACE,A,XSTYLE=4,YSTYLE=4,ZSTYLE=4,/NOERASE
   ```

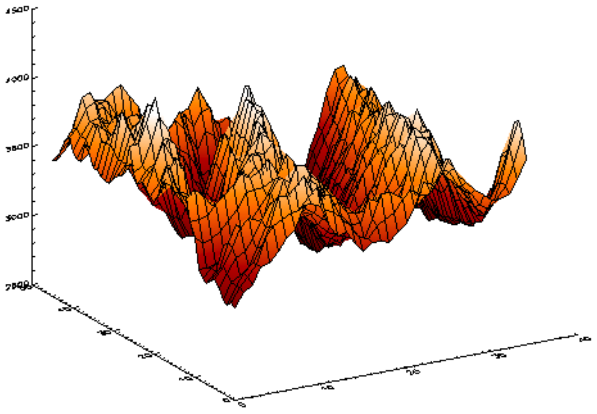


Figure 7-6: Byte-scaled surface plot with an overlaid wire-frame

Displaying a Contour

Another way to view a two-dimensional array is as a contour plot. A simple contour plot of the Data can be created.

1. Set the array size back to its original 350 by 450 size by entering:

```
A=MARBELLS_BINARY.A
```

2. Plot the contour:

```
CONTOUR, A
```

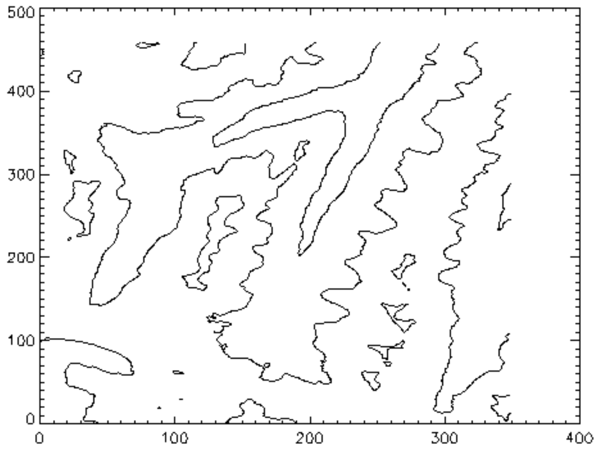


Figure 7-7: Contour plot

That command was very simple, but the resulting plot was not as informative as it could be.

3. Create a customized CONTOUR plot with more elevations and labels by entering:

```
CONTOUR,A,NLEVELS=8,C_LABELS=[0,1]
```

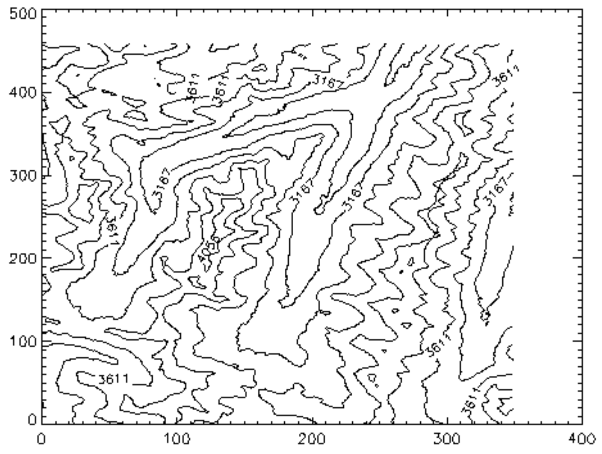


Figure 7-8: Contour plot with elevation labeled

By using the NLEVELS keyword, CONTOUR was told to plot eight equally-spaced elevation levels. The C_LABELS keyword specifies which contour levels should be labeled. By default, every other contour is labeled. C_LABELS allows you to override this default and explicitly specify the levels to label.

4. Similarly, you can create a filled contour plot where each contour level is filled with a different color (or shade of gray) by setting the FILL keyword. To do this, enter:

```
CONTOUR,A,NLEVELS=8,/FILL
```

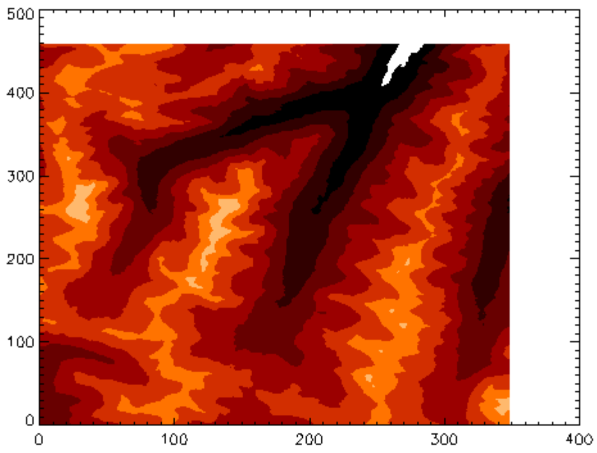


Figure 7-9: Contour plot with filled contour plot

5. To outline the resulting contours, make another call to CONTOUR and set the OVERPLOT keyword to keep the previous plot from being erased.

   You can add tickmarks that indicate the slope of the contours (the tickmarks point in the downhill direction) by setting the DOWNHILL keyword:

```
CONTOUR,A,NLEVELS=8,/OVERPLOT,/DOWNHILL
```

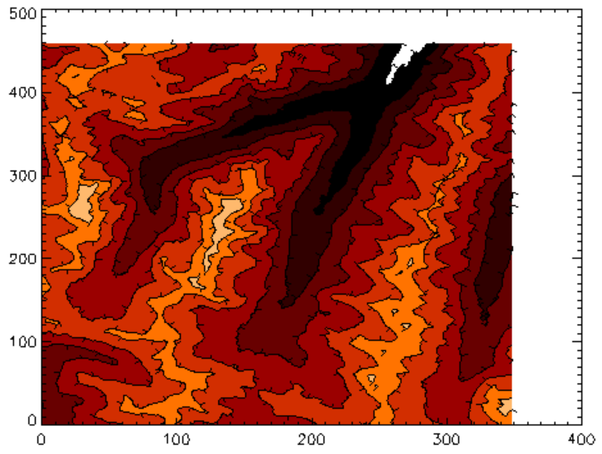


Figure 7-10: Contour plot with downhill tickmarks labeled

6. CONTOUR plots can be rendered from a three-dimensional perspective.

   First, set up the default 3-D viewing angle by entering:

   ```
   SURFR
   ```

7. By using the T3D keyword in the next call to CONTOUR, the contours will be drawn as seen from a 3-D perspective. Enter:

   ```
   CONTOUR,A,NLEVELS=8,/T3D
   ```

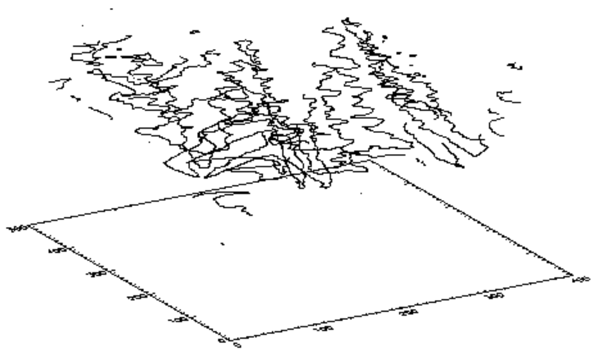


Figure 7-11: Contour plot with 3-D contour plot

Plotting with SHOW3

In addition to IDL’s built-in routines, there are many functions and procedures included with IDL that are written in the IDL language and that can be changed, customized, or even rewritten by IDL users. The SHOW3 procedure is one of these routines.

1. Create a plot that shows a two-dimensional array as an image, wire-frame surface, and contour simultaneously.

```
SHOW3,A
```

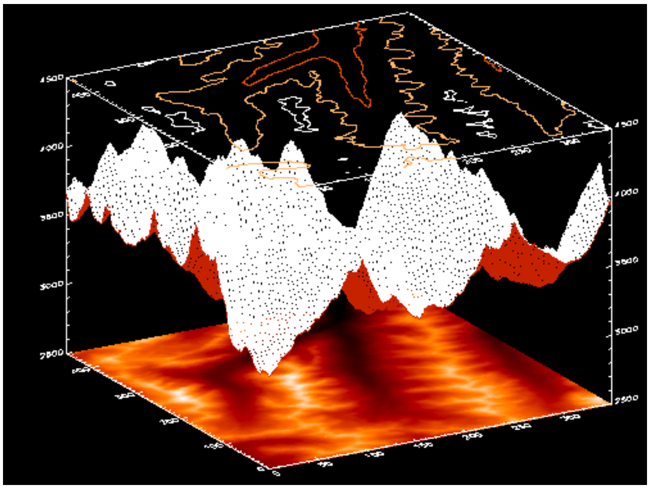


Figure 7-12: Combined surface and contour plots using SHOW3

Using LIVE_SURFACE for Plotting

The LIVE_SURFACE procedure allows interactive manipulation using the mouse and keyboard. Usually, LIVE_SURFACE is most suitable for relatively small data sets since the interactive environment requires extra system resources.

After you execute LIVE_SURFACE, you can double-click on a section of the surface to display a properties dialog. The buttons in the upper left of the image window allow you many options (print, undo, redo, copy, line, rectangle, text and so on).

1. Here, to visualize a surface representation, enter the following:

```
LIVE_SURFACE,A
```

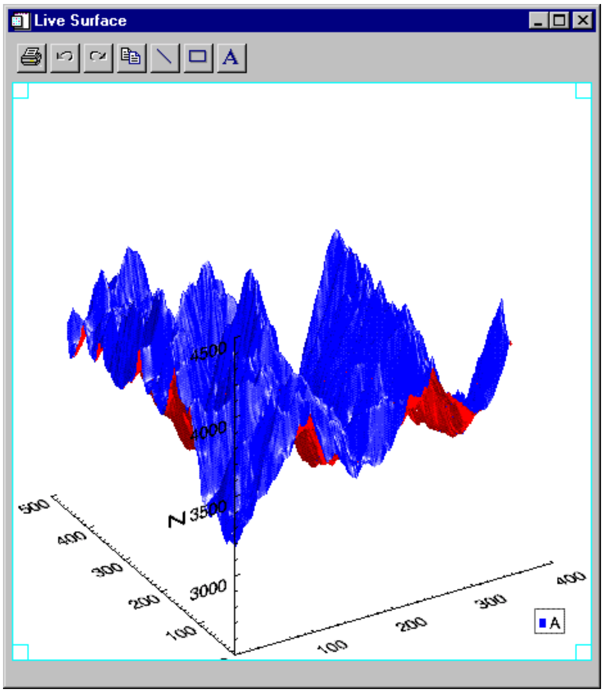
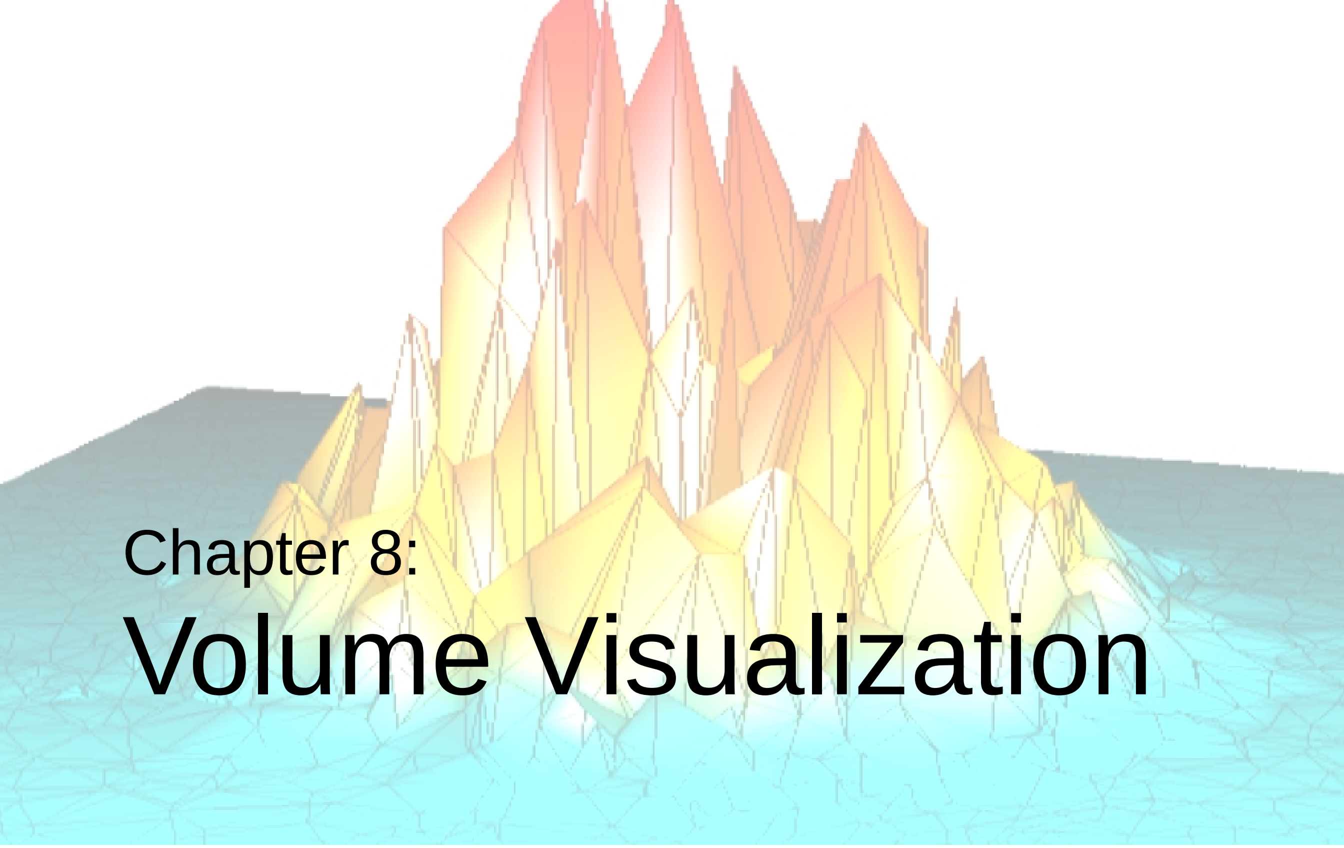


Figure 7-13: Visualization of surface representation using LIVE_SURFACE.

More Information on 3-D Plotting

The SURFACE, CONTOUR, and SHADE_SURF commands have many more options that can be used to create even more complex, customized plots. For more information on plotting multi-dimensional arrays, see Chapter 18, “Plotting Multi-Dimensional Arrays” in *Using IDL* and the documentation for specific routines in the *IDL Reference Guide*.



Chapter 8: Volume Visualization

This chapter describes the following topics:

IDL and Volume Visualization 100
Reading in a Dataset for Visualization . . . 101
3-D Transformations 102
Visualizing an Iso-Surface 104
Making Slices with the IDL Slicer 106
Displaying a Surface with the Slicer 108
More Information on Volume Visualization 110

IDL and Volume Visualization

IDL can be used to visualize multi-dimensional volume datasets. Given a 3-D grid of density measurements, IDL can display a shaded surface representation of a constant-density surface (also called an iso-surface). For example, in medical imaging applications, a series of 2-D images can be created by computed tomography or magnetic resonance imaging. When stacked, these images create a grid of density measurements that can be contoured to display the surfaces of anatomical structures.

This chapter demonstrates the use of the SHADE_VOLUME and POLYSHADE commands for iso-surface visualization.

Note

To simplify obtaining useful results from the examples in this manual, create a bitmap buffer for your graphic windows by entering the following command at the IDL command prompt:

```
DEVICE, RETAIN=2, DECOMPOSED=0
```

Reading in a Dataset for Visualization

The following steps illustrate the use of BINARY_TEMPLATE and READ_BINARY to read in a dataset:

1. First, create the template for reading the data. At the IDL Command Line, enter:

   ```
   MYTEMPLATE=BINARY_TEMPLATE(FILEPATH('head.dat',$
      SUBDIR=['examples', 'data']))
   ```

   The binary template dialog box appears.

2. In the Template Name field, enter “Head” as the name of the new template.
3. Now we are ready to enter the field values, click the box in the lower left corner of the dialog box called “New Field”.
4. When the New/Modify Field dialog appears, enter “B” as the field name. Check the box in the upper right corner marked “Returned” since we will want our data set returned at the time it is read.

   At the “Number of Dimensions” pull-down menu, be sure to specify that we are dealing with a three-dimensional data set. These data are contained in an 80 by 100 by 57 array, so we will enter these values in the three boxes marked “size”.

   Finally, let the binary template dialog know that we are dealing with Byte type data by specifying “Byte (Unsigned 8-bits) at the “Type” pull-down menu.

   Once you have entered the above data, the binary template dialog appears once again showing the specifications you have made.

5. Click “OK”.

   Now we will use the READ_BINARY procedure to read the data defined by template we have just created.

6. At the IDL Command Line, enter:

   ```
   HEAD_BINARY=READ_BINARY(FILEPATH('head.dat',SUBDIR=['example
   s',$
      'data']),TEMPLATE=MYTEMPLATE)
   ```

3-D Transformations

When creating “3-D” plots (for example, surfaces, shaded surfaces, and volume visualizations), a three-dimensional transformation needs to be set up. The 3-D transformation applies the requested translation, rotation, and scaling to a 3-D plot before displaying it.

Three-dimensional transformations are especially important when using the POLYSHADE routine. Unless the transformation is set up such that the entire volume is visible, the volume will not be rendered correctly. Once a 3-D transformation has been established, most IDL plotting routines can be made to use it by including the T3D keyword.

There are a number of ways to set up a transformation matrix in IDL.

One way is that a transformation matrix can be entered explicitly into the system variable !P.T. This method is rather difficult, because you have to figure out the transformation yourself. More information about the transformation matrix can be found in Chapter 18, “Plotting Multi-Dimensional Arrays” of *Using IDL*.

Another method, is the SURFACE and SHADE_SURF commands, which automatically create a 3-D transformation based on the datasets being visualized.

1. For example, specify a slice of the data:

   ```
   SLICE=(HEAD_BINARY.B)[*,*,25]
   ```

2. Now surface the slice specified:

   ```
   SURFACE, SLICE
   ```

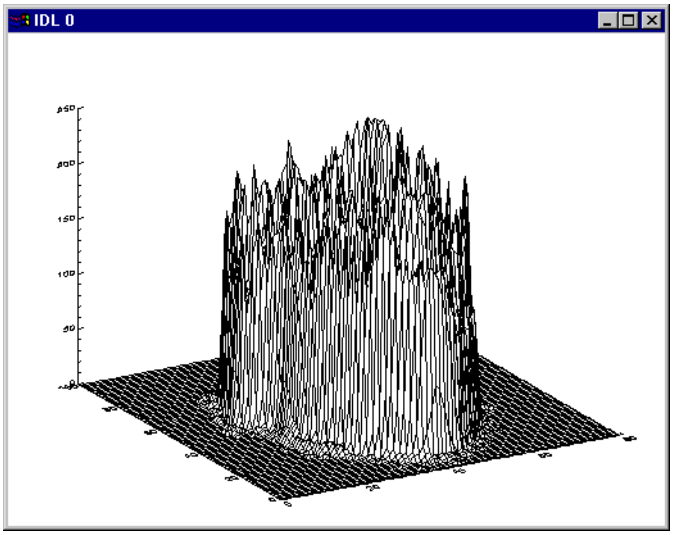


Figure 8-1: Using SURFACE to visualize a slice of head.dat

A number of different IDL procedures that simplify the creation of 3-D transformations can be used. Keyword arguments to some of these procedures allow you to set viewing angles and data ranges. The procedures then create the appropriate transformation matrix for you and store it in !P.T. These procedures include T3D, SCALE3, SCALE3D, and SURFR. For more information on these routines, consult the *IDL Reference Guide*.

Visualizing an Iso-Surface

Two IDL commands, SHADE_VOLUME and POLYSHADE, are used together to visualize an iso-surface. SHADE_VOLUME generates a list of polygons that define a 3-D surface given a volume dataset and a contour (or *density*) level. The function POLYSHADE can then be used to create a shaded-surface representation of the iso-surface from those polygons.

Like many other IDL commands, POLYSHADE accepts the T3D keyword that makes POLYSHADE use a user-defined 3D transformation. Before you can use POLYSHADE to render the final image, you need to set up an appropriate three-dimensional transformation. The XRANGE, YRANGE, and ZRANGE keywords accept 2-element vectors, representing the minimum and maximum axis values, as arguments.The POLYSHADE function returns an image based upon the list of vertices, `V`, and list of polygons, `P`. The T3D keyword tells POLYSHADE to use the previously-defined 3D transformation. The TV procedure displays the shaded-surface image.

Enter the following lines:

1. Create the polygons and vertices that define the iso-surface with a value of 70. Return the vertices in V and the polygons in P:

   ```
   SHADE_VOLUME,HEAD_BINARY.B,70,V,P,/LOW
   ```

2. Set appropriate limits for the X, Y, and Z axes with the SCALE3 procedure:

   ```
   SCALE3,XRANGE=[0,80],YRANGE=[0,100],ZRANGE=[0,57]
   ```

3. Display a shaded-surface representation of the previously generated arrays of vertices and polygons:

```
TV,POLYSHADE(V,P,/T3D)
```

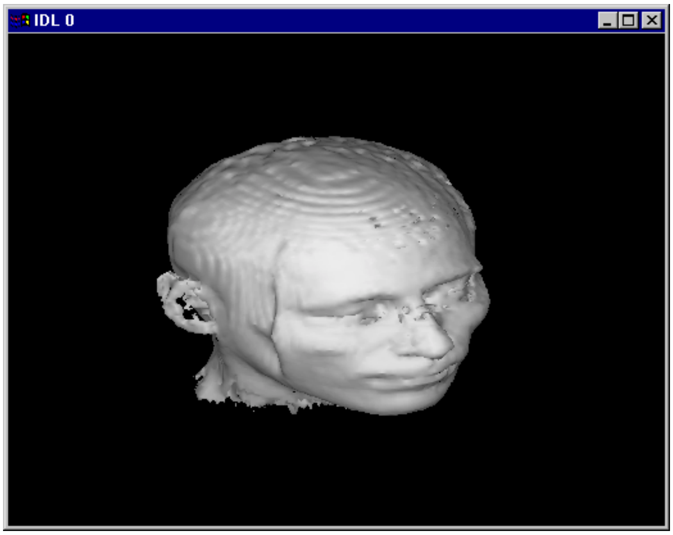


Figure 8-2: Shaded-surface representation using POLYSHADE

Making Slices with the IDL Slicer

Another useful volume visualization tool is IDL’s SLICER3 procedure. The Slicer is a widget-based application that allows you to create iso-surfaces and pass cutting planes through 3-D datasets.

The IDL Slicer provides many other volume visualization techniques. As the name implies, the slicer allows you to look at slices through a volume dataset.

1. To use the slicer with dataset B, it is first required to pass in a pointer to the data set by entering the following at the IDL Command Line:

   ```
   BDATA=PTR_NEW(HEAD_BINARY.B)
   ```

2. Then enter:

   ```
   SLICER3,BDATA
   ```

3. The IDL Slicer appears. The Slicer window will come up empty by default though the data is loaded. Be sure that the **Mode** pull-down menu is set to **Slice** which is the default. Position the pointer within the cube. Hold down the left mouse button and move the mouse. (In *IDL for Macintosh*, the mouse button is interpreted as the left mouse button.) An outline of the cutting plane appears. This plane moves only in the direction indicated by the orientation display.

Move the cutting plane to the center of the volume and release the mouse button. A cross-section of the volume is displayed.

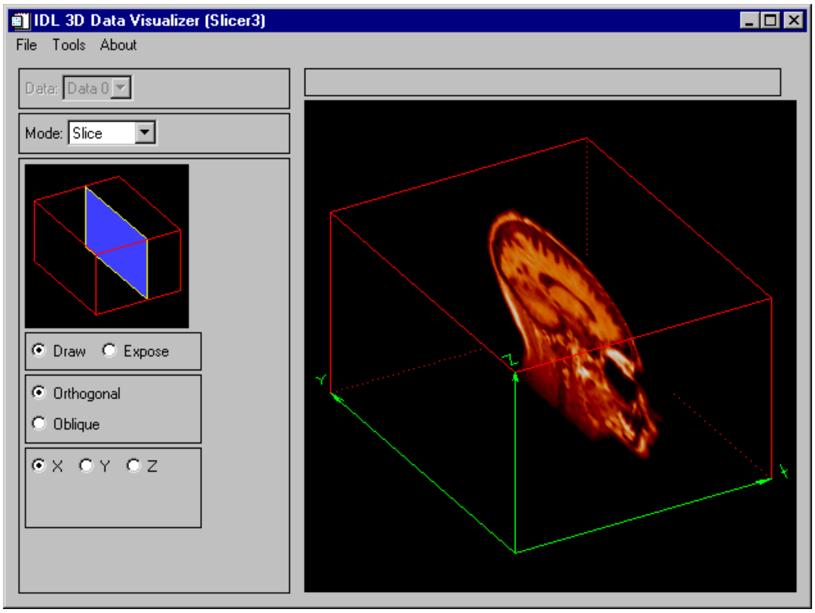


Figure 8-3: IDL Slicer3

4. To make slices in different orientations, move the cursor into the large drawing window and press the right mouse button.
5. To simulate a right mouse button press, *IDL for Macintosh* users can hold down the command key and click the mouse. The orientation display changes to show the new direction of the cutting plane.
6. Click the right button a second time to see the third possible orientation.
7. Make slices in these orientations by clicking on the mouse button and dragging the cutting plane outline to the desired location.

Displaying a Surface with the Slicer

To display a surface with the IDL Slicer, do the following:

1. To create a surface in Slicer similar to the one you created previously at the IDL command line, click on the **Surface** option on the **Mode** pull-down menu on the Slicer. A **Surface Threshold** window, a slider, and a number of new buttons should appear.
2. Click in the **Surface Threshold** and slide the determiner line to choose the **Display** button. A status window reports on the number of vertices and polygons generated and then the iso-surface appears.

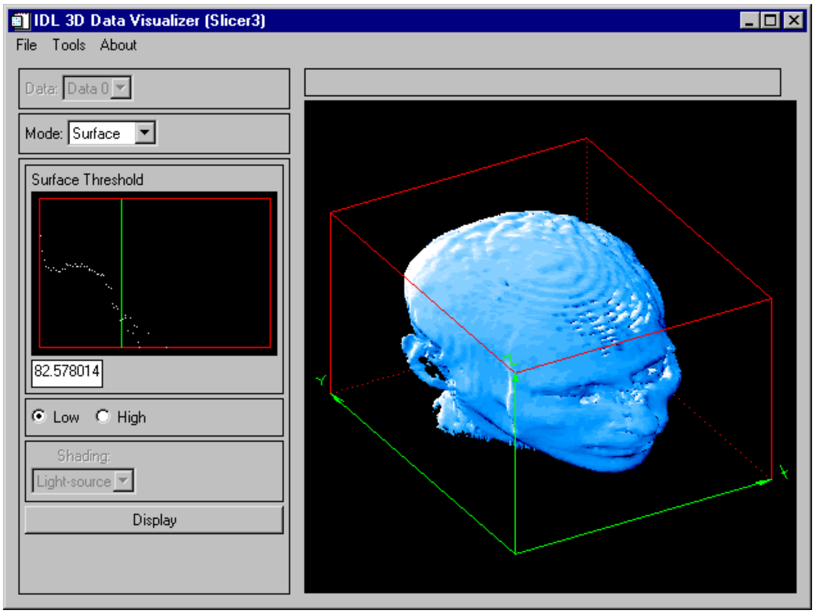


Figure 8-4: IDL Slicer3 with a surface

Dismiss the Slicer and Volume Windows

When you are done experimenting with the Slicer, before continuing with other chapters in this book, you should dismiss the Slicer window.

1. To exit the volume window, enter the following at the Command Line:

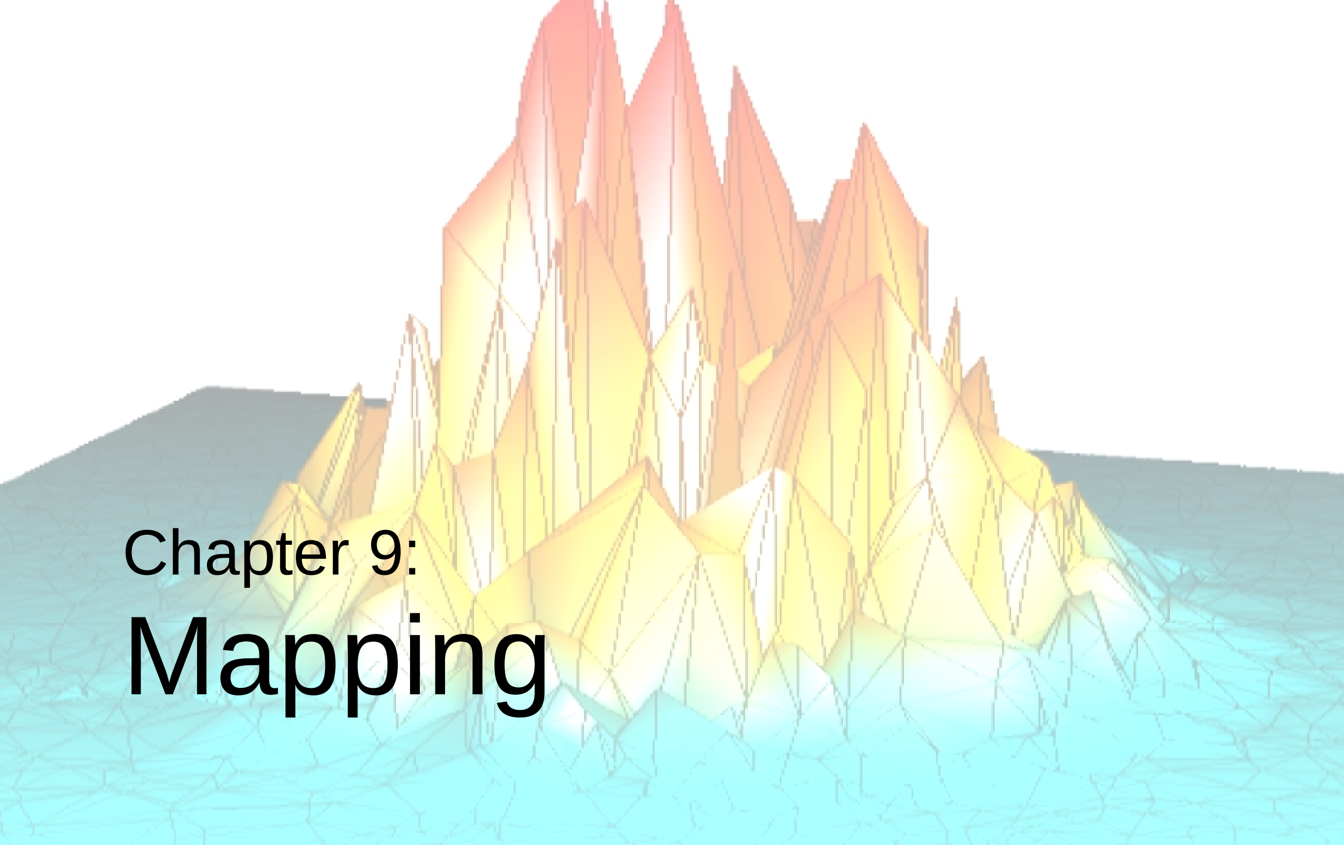
   ```
   WDELETE
   ```

2. To exit the IDL Slicer, choose the path **File** → **Quit**.

More Information on Volume Visualization

More information on the SHADE_VOLUME procedure can be found in Chapter 18, “Plotting Multi-Dimensional Arrays” of *Using IDL*. Also, see SCALE3, SHADE_VOLUME, and TV in the *IDL Reference Guide*.

A complete description of the slicer’s capabilities is beyond the scope of this tutorial. Click the Slicer’s **Help** button or see SLICER3 in the *IDL Reference Guide* for more information.



Chapter 9: Mapping

This chapter describes the following topics:

IDL and Mapping 112
Drawing Map Projections 113
Drawing an Orthographic Projection 115
Plotting a Portion of the Globe 116
Plotting Data on Maps 117
Reading Latitudes and Longitudes 119
Plotting Contours Over Maps 120
Warping Images to Maps 122
More Information on Mapping 127

IDL and Mapping

IDL's mapping facilities allow you to plot data over different projections of the globe. This chapter shows how to display various map projections and plot data over them.

In this chapter, we will enter statements at the IDL Command Input Line. This demonstrates IDL's interactive capability, and shows how easy it is to manipulate your data.

Note

To simplify obtaining useful results from the examples in this manual, create a bitmap buffer for your graphic windows and to use a maximum of 256 colors by entering the following command at the IDL command prompt:

```
DEVICE, RETAIN=2, DECOMPOSED=0
```

Drawing Map Projections

Drawing continental outlines and plotting data in different projections is easy using IDL’s mapping routines. The MAP_SET routine is the heart of the mapping package. It controls the type of projection and the limits of the global region to be mapped.

1. Reset the graphics window to default size:

```
WINDOW
```

2. Display a cylindrical projection map of the world:

```
MAP_SET,/CYLINDRICAL,/GRID,/CONTINENTS,/LABEL
```

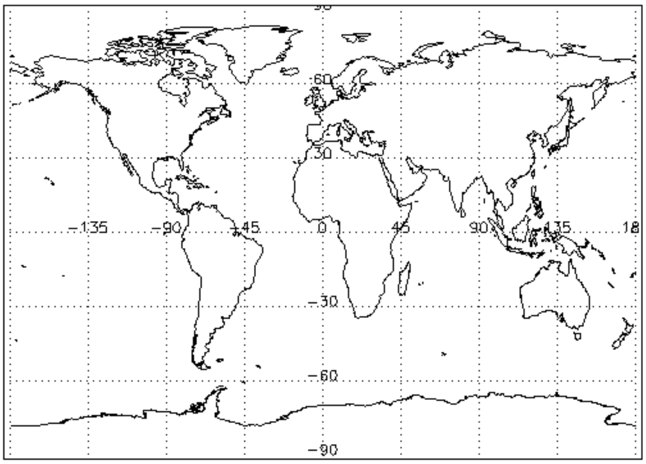


Figure 9-1: A cylindrical projection

The CYLINDRICAL keyword tells MAP_SET to use the cylindrical projection. The GRID keyword causes the latitude and longitude lines to be drawn. The LABEL keyword adds the latitude and longitude labels. The CONTINENTS keyword tells MAP_SET to draw continental outlines.

A similar map could be created by entering a series of separate commands to set up the type of projection, draw the continent outlines, and then draw the grid lines.

Although the single-line MAP_SET command is quicker to enter, by using the separate MAP_SET, MAP_GRID, and MAP_CONTINENTS commands, you exercise more control over the map colors, fills, and so on.

3. Load a new color table.

```
LOADCT,39
```

4. Display a Miller cylindrical projection of the world.

```
MAP_SET,/MILLER
```

5. Draw the continent outlines. The FILL keyword fills in the continents using the color specified by the COLOR keyword.

```
MAP_CONTINENTS,COLOR=220,/FILL
```

6. Draw the grid lines. The COLOR keyword specifies the color of the grid lines. The LABEL keyword labels the lines.

```
MAP_GRID,COLOR=160,/LABEL
```

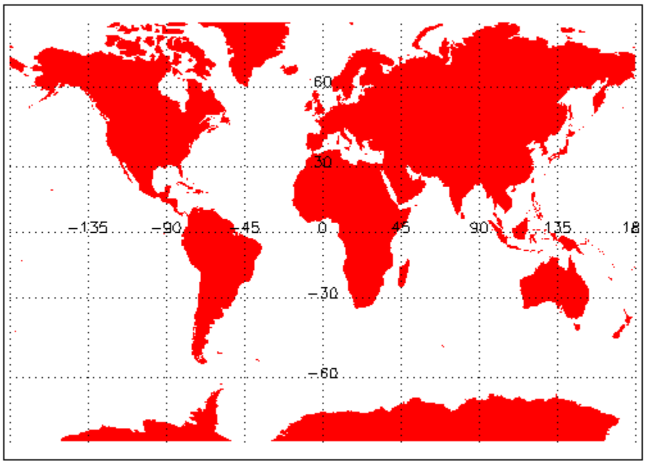


Figure 9-2: Miller cylindrical projection with MAP_CONTINENTS and MAP_GRID

The order of MAP_GRID and MAP_CONTINENTS depends on how you wish to display your map. In the above example, if you call MAP_GRID before MAP_CONTINENTS, the filled continents are drawn over the labeled grid lines.

7. Dismiss the graphics window:

```
WDELETE
```

Drawing an Orthographic Projection

To draw a map that looks more like a globe, use the orthographic projection.

1. Open a graphics window for viewing:

```
WINDOW
```

2. Enter the following at the Command Line:

```
MAP_SET,30,-100,0,/ORTHOGRAPHIC,/ISOTROPIC,/GRID, $
   /CONTINENTS,/LABEL,/HORIZON
```

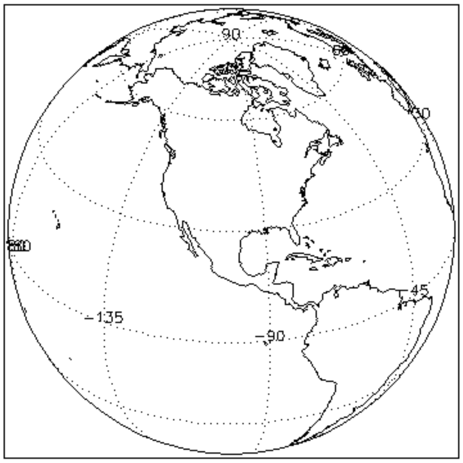


Figure 9-3: Orthographic projection showing North America at the center

The numbers following the MAP_SET command (30, -100, and 0) are the latitude and longitude to be centered and the angle of rotation for the North direction. The ISOTROPIC keyword creates a map that has the same scale in the vertical and horizontal directions, so we get a circular map in a rectangular window. IDL keywords (but not function and procedure names) can always be abbreviated to their minimum unique length. The GRID, COLOR, and LABEL keywords work the same as before. The HORIZON keyword draws the line at which the horizon exists. Without the HORIZON keyword, MAP_SET only draws the grid and the continents.

3. Dismiss the graphics window:

```
WDELETE
```

Plotting a Portion of the Globe

You do not always have to plot the entire globe, you can plot a section of the globe by using the LIMIT keyword which specifies a region of the globe to plot.

1. Open a graphics window for viewing:

```
WINDOW
```

2. Enter the following at the Command Line:

```
MAP_SET,32,-100,/AZIM,LIMIT=[10, -130, 55, -70], $
   /GRID,/CONT,/LABEL
```

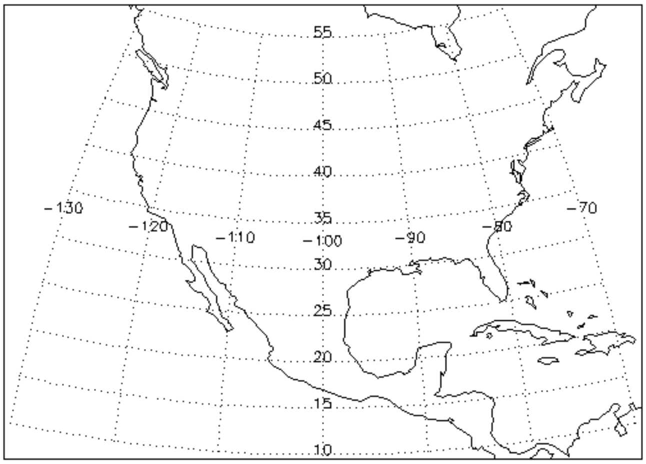


Figure 9-4: Azimuthal equidistant projection

The azimuthal equidistant projection shows the United States and Mexico. The AZIM keyword selects the azimuthal equidistant projection. The keyword LIMIT is set equal to a four-element vector containing the minimum latitude, minimum longitude, maximum latitude, and maximum longitude.

3. Dismiss the graphics window:

```
WDELETE
```

Plotting Data on Maps

You can annotate plots easily in IDL. To plot the location of selected cities in North America, as shown in the following figure, you need to create three arrays: one to hold latitudes, one to hold longitudes, and one to hold the names of the cities being plotted.

1. Open a graphics window for viewing:

   ```
   WINDOW
   ```

2. Create a 5-element array of floating-point values representing latitudes in degrees North of zero.

   ```
   LATS=[40.02,34.00,38.55,48.25,17.29]
   ```

3. The values in LONS are negative because they represent degrees West of zero longitude.

   ```
   LONS=[-105.16,-119.40,-77.00,-114.21,-88.10]
   ```

4. Create a five-element array of string values. Text strings can be enclosed in either single quotes ('text') or double quotes ("text").

   ```
   CITIES=['Boulder, CO','Santa Cruz, CA',$
   'Washington, DC','Whitefish, MT','Belize, Belize']
   ```

5. Draw a Mercator projection featuring the United States and Mexico.

   ```
   MAP_SET,/MERCATOR,/GRID,/CONTINENT,LIMIT=[10,-130,60,-70]
   ```

6. Place a plotting symbol at the location of each city.

   ```
   PLOTS,LONS,LATS,PSYM=4,SYMSIZE=1.4,COLOR=220
   ```

7. Place the names of the cities near their respective symbols.

```
XYOUTS,LONS,LATS,CITIES,COLOR=80, $
   CHARTHICK=2,CHARSIZE=1.25,ALIGN=0.5
```

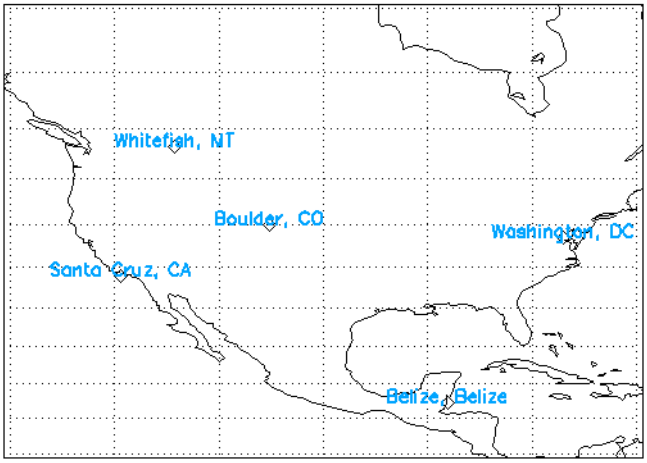


Figure 9-5: Annotating a map projection

The PSYM keyword makes PLOTS use diamond-shaped plotting symbols instead of connecting lines. The SYMSIZE keyword controls the size of the plotting symbols. XYOUTS draws the characters for each element of the array CITIES at the corresponding location specified by the array elements of LONS and LATS. The CHARTHICK keyword controls the thickness of the text characters and the CHARSIZE keyword controls their size (1.0 is the default size). Setting the ALIGN keyword to 0.5 centers the city names over their corresponding data points.

Reading Latitudes and Longitudes

If a map projection is displayed, IDL can return the position of the cursor over the map in latitude and longitude coordinates.

1. Enter the command:

```
CURSOR, LON, LAT & PRINT, LAT, LON
```

The CURSOR command reads the “X” and “Y” positions of the cursor when the mouse button is pressed and returns those values in the LON and LAT variables. Use the mouse to move the cursor over the map window and click on any point. The latitude and longitude of that point on the map are printed in the Output Log.

2. When you are finished with your map, dismiss the graphics window:

```
WDELETE
```

Plotting Contours Over Maps

Contour plots can easily be drawn over map projections by using the OVERPLOT keyword to the CONTOUR routine. See the map in the figure below. Enter the following at the Command Line:

1. Open a graphics window for viewing:

   ```
   WINDOW
   ```

2. Create a dataset to plot:

   ```
   A=DIST(91)
   ```

3. Create an X value vector containing 91 values that range from -90 to 90 in 2 degree increments:

   ```
   LAT=FINDGEN(91)*2-90
   ```

4. Create a Y value vector containing 91 values that range from -180 to 180 in 4 degree increments:

   ```
   LON=FINDGEN(91)*4-180
   ```

5. Create a new sinusoidal map projection over which to plot the data:

   ```
   MAP_SET,/GRID,/CONTINENTS,/SINUSOIDAL,/HORIZON
   ```

6. Draw a twelve-level contour plot of array A over the map:

   ```
   CONTOUR,A,LON,LAT,/OVERPLOT,NLEVELS=12
   ```

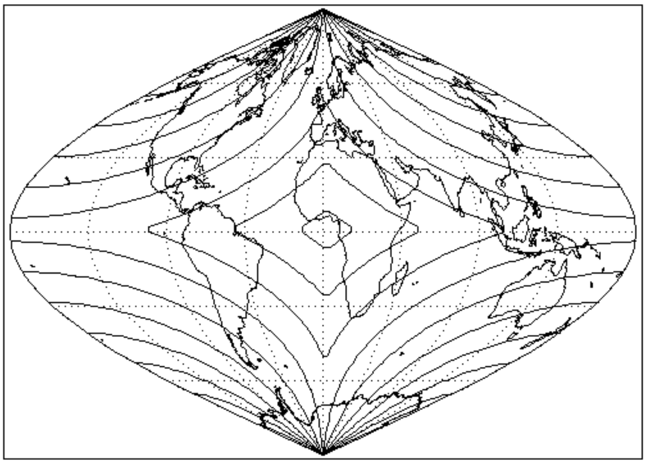


Figure 9-6: Plotting contours over maps

Since latitudes range from -90 to 90 degrees and longitudes range from -180 to 180 degrees, you created two vectors containing the “X” and “Y” values for CONTOUR to use in displaying the array A. If the X and Y values are not explicitly specified, CONTOUR will plot the array A over only a small portion of the globe.

7. When you are finished with the map, dismiss the graphics window:

```
WDELETE
```

Warping Images to Maps

Image data can also be displayed on maps. The MAP_IMAGE function returns a warped version of an original image that can be displayed over a map projection. In this example, elevation data for the entire globe is displayed as an image with continent outlines and grid lines overlaid.

1. Define the template for the file worldelv.dat. This file contains a 360 by 360 square array of byte values.

```
WORLDTEMPLATE=BINARY_TEMPLATE(FILEPATH('worldelv.dat', $
   SUBDIR=['examples', 'data']))
```

2. When the binary template dialog box appears, name the template “WORLDTEMPLATE” and then click **New Field**.
3. In the **New Field** dialog, enter “W” for the **Field Name**, be sure to specify that you have two dimensions and that the field sizes are 360 and 360.
4. Also select **Byte (unsigned)** in the **Type** field. Now click **OK** in the **New Field** dialog. Click “OK” to close the binary template dialog as well. Next, read the file by entering:

```
WORLDELV_BINARY=READ_BINARY(FILEPATH('worldelv.dat', $
   SUBDIR=['examples', 'data']),TEMPLATE=WORLDTEMPLATE)
```

5. Load a color table.

```
loadct, 26
```

6. View the data as an image.

```
TV, WORLDELV_BINARY.W
```

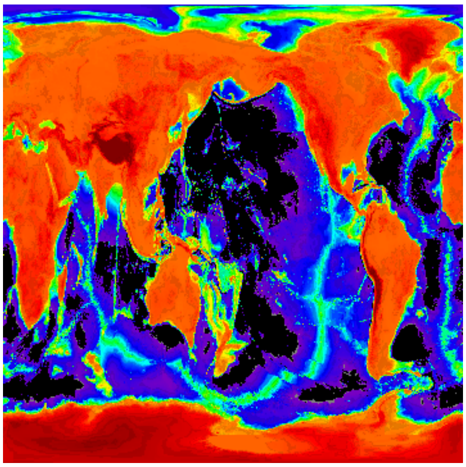


Figure 9-7: worldelv.dat visualized with TV

The first column of data in this image corresponds to 0 degrees longitude. Because MAP_IMAGE assumes that the first column of the image being warped corresponds to -180 degrees, we'll use the SHIFT function on the dataset before proceeding.

7. Shift the array 180 elements in the row direction and 0 elements in the column direction to make -180 degrees the first column in the array.

```
WORLDELV_BINARY.W=SHIFT(WORLDELV_BINARY.W, 180, 0)
```

8. View the data as an image.

```
TV, WORLDELV_BINARY.W
```

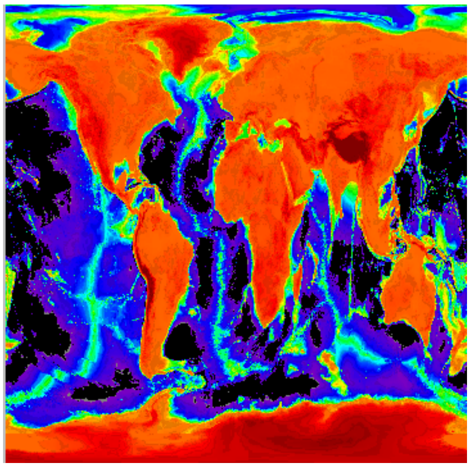


Figure 9-8: Shifting the array

From the image contained in the data, we can create a warped image to fit any of the available map projections. A map projection must be defined before using MAP_IMAGE, because MAP_IMAGE uses the currently defined map parameters.

9. Create a Mollweide projection with continents and gridlines.

```
MAP_SET,/MOLLWEIDE,/CONT,/GRID,COLOR=100
```

10. Warp the image using bilinear interpolation and save the result in the variable new.

```
NEW=MAP_IMAGE(WORLDELV_BINARY.W,SX,SY,/BILIN)
```

The SX and SY in the command above are output variables that contain the X and Y position at which the image should be displayed. Setting the BILIN keyword causes bilinear interpolation to be used, resulting in a smoother warped image.

11. Display the new image over the map:

```
TV,NEW,SX,SY
```

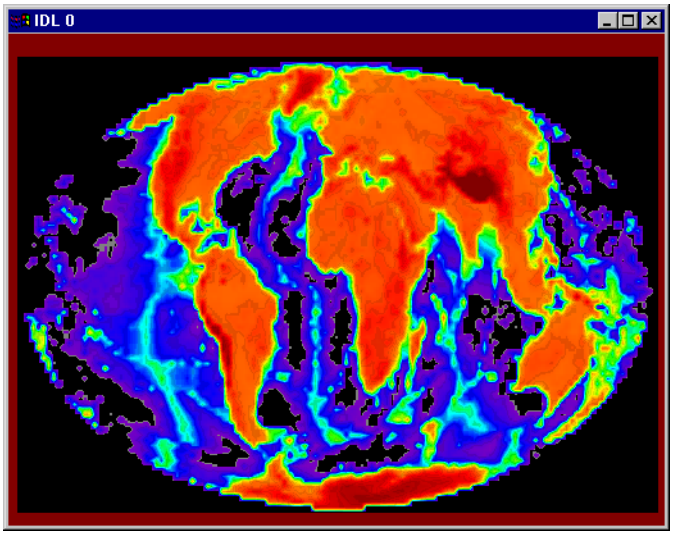


Figure 9-9: Warping an image to a map

The SX and SY variables provide TV with the proper starting coordinates for the warped image. TV usually displays images starting at position (0, 0). See the map in the previous figure. Note that the warped image gets displayed over the existing continent and grid lines.

12. The continent outlines and thick grid lines can be displayed, as shown next, by entering:

```
MAP_CONTINENTS
MAP_GRID,GLINETHICK=3
```

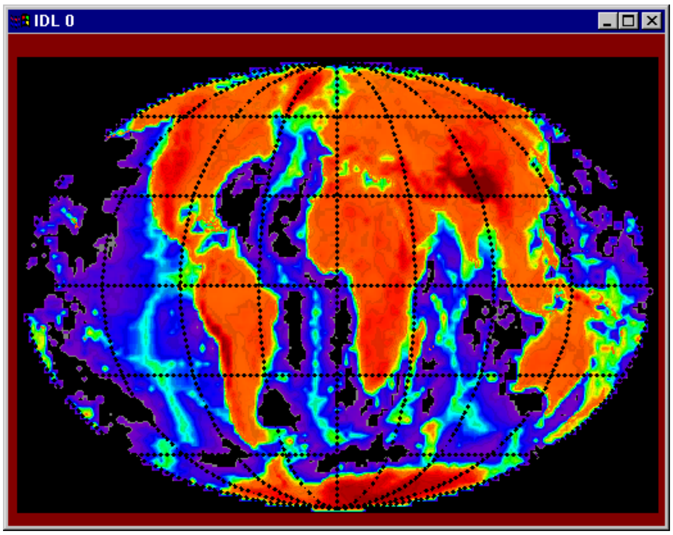
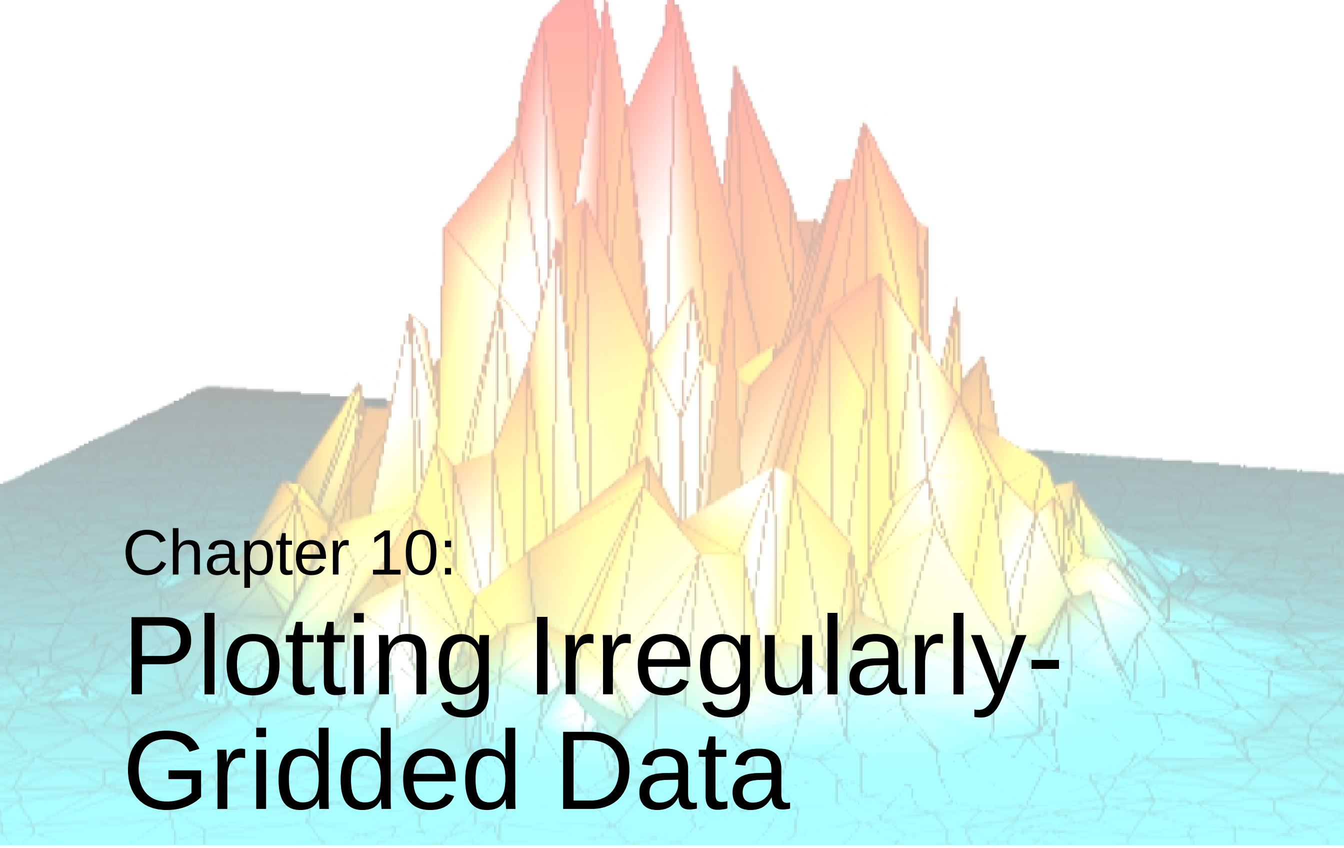


Figure 9-10: Showing gridlines and continents

More Information on Mapping

More information on the IDL mapping routines can be found in *Using IDL* and in the *IDL Reference Guide*.



Chapter 10:

Plotting Irregularly-Gridded Data

This chapter describes the following topics:

IDL and Plotting Irregularly-Gridded Data 130
Creating a Dataset 131
The TRIANGULATE Procedure 133
Plotting the Results with TRIGRID 135
More Information on Gridding 137

IDL and Plotting Irregularly-Gridded Data

IDL can be used to display and analyze irregularly-gridded data. IDL routines allow you to easily fit irregularly-sampled data to a regular grid. This regularly-gridded data can then be sent to IDL's plotting routines. In this chapter, we will see how easy it is to manipulate your irregularly-gridded data.

Note

To simplify obtaining useful results from the examples in this manual, create a bitmap buffer for your graphic windows and to use a maximum of 256 colors by entering the following command at the IDL command prompt:

```
DEVICE, RETAIN=2, DECOMPOSED=0
```

Creating a Dataset

Create a set of 32 irregularly-gridded data points in 3-D space that we can use as arguments to the TRIGRID and TRIANGULATE functions.

1. Open a window for viewing:

   ```
   WINDOW
   ```

2. Set SEED to the longword value 1. SEED is used to generate random points.

   ```
   SEED=1L
   ```

3. Set the number of points to be randomly generated.

   ```
   N=32
   ```

4. Create a set of X values for each of the 32 data points.

   ```
   X=RANDOMU(SEED,N)
   ```

5. Create a set of Y values for each of the 32 data points.

   ```
   Y=RANDOMU(SEED,N)
   ```

6. Create a set of Z values for each of the 32 data points from the X and Y values.

   ```
   Z=EXP(-3*((X-0.5)^2+(Y-0.5)^2))
   ```

7. Plot the XY positions of the random points.

```
PLOT,X,Y,PSYM=1,TITLE='Random XY Points'
```

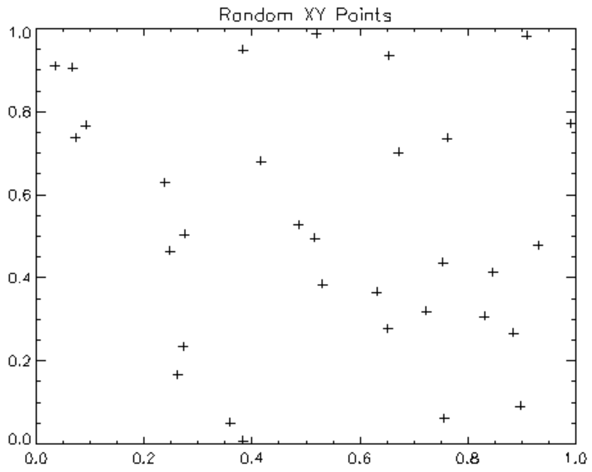


Figure 10-1: Plot of random values

8. Dismiss the graphics window:

```
WDELETE
```

The TRIANGULATE Procedure

The TRIANGULATE procedure constructs a Delaunay triangulation of a planar set of points. After a triangulation has been found for a set of irregularly-gridded data points, the TRIGRID function can be used to interpolate surface values to a regular grid.

1. Open a window for viewing:

   ```
   WINDOW
   ```

2. To return a triangulation in the variable TR, enter the command:

   ```
   TRIANGULATE,X,Y,TR
   ```

   The variable TR now contains a three-element by 54-element longword array (you may see this by typing “`Help, TR`” at the IDL Command Line).

Note

This is not always a 54-element array, it may vary based on the number of points.

3. To produce a plot of the triangulation, shown below, enter the following commands:

   ```
   PLOT,X,Y,PSYM=1,TITLE='Triangulation'

   FOR i=0,N_ELEMENTS(TR)/3 - 1 DO BEGIN & T= $
      [TR[*, i],TR[0, i]] & PLOTS,X[T],Y[T] & ENDFOR
   ```

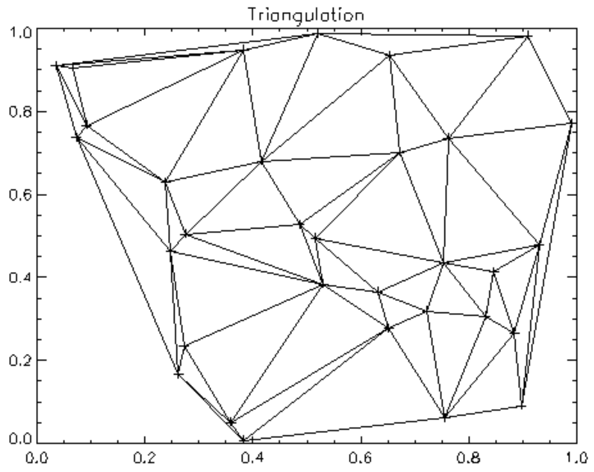


Figure 10-2: The triangulation of the random values

4. Dismiss the graphics window

```
WDELETE
```

Plotting the Results with TRIGRID

Now that we have the triangulation TR, the TRIGRID function can be used to return a regular grid of interpolated Z values.

1. Display a surface plot of the gridded data by passing the result of the TRIGRID function to SURFACE, using the default interpolation technique and add a title to the plot, shown below, by entering:

```
SURFACE,TRIGRID(X,Y,Z,TR)
XYOUTS,.5,.9,'Linear Interpolation',ALIGN=.5,/NORMAL
```

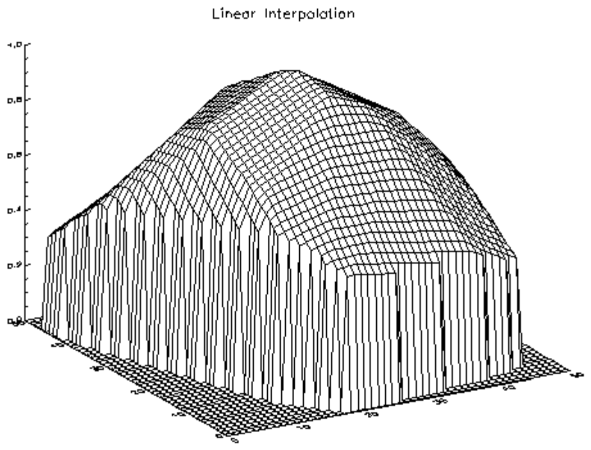


Figure 10-3: Linear interpolation of triangulated data

The TRIGRID function can also return a smoothed interpolation. Set the QUINTIC keyword to use a quintic polynomial method when interpolating the grid.

2. Display the results of the quintic gridding method, shown below, by entering:

```
SURFACE,TRIGRID(X,Y,Z,TR,/QUINTIC XYOUTS, .5,.9,’Quintic
Interpolation',ALIGN=.5,/NORMAL
```

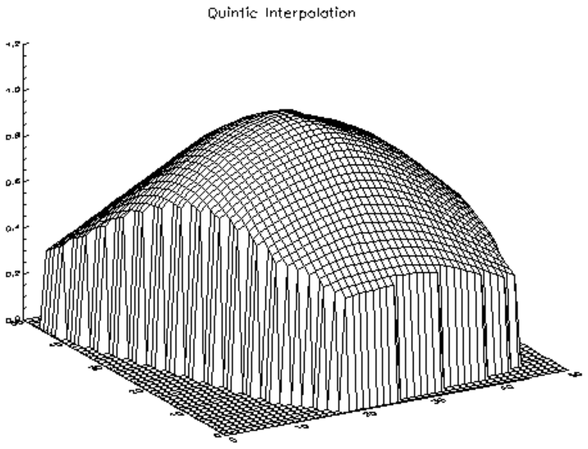
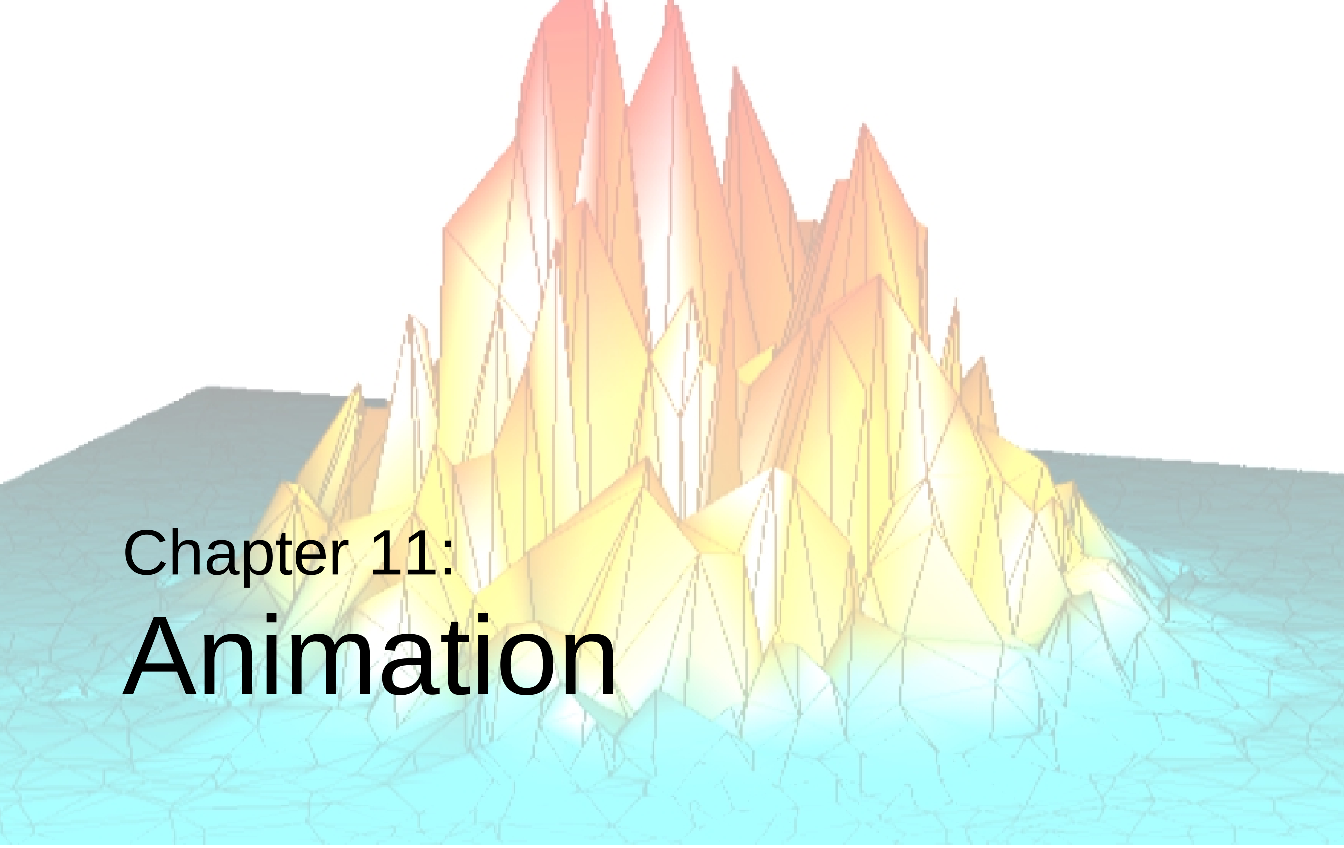


Figure 10-4: Quintic interpolation of triangulated data

More Information on Gridding

More information on the TRIGRID and TRIANGULATE routines as well as other triangulation routines, can be found in the *IDL Reference Guide*.



Chapter 11: Animation

This chapter describes the following topics:

IDL and Animation 140
Animating a Series of Images 141
Displaying an Animation as a Wire Mesh . 143
Animation with XINTERANIMATE 146
Cleaning Up the Animation Windows . . . 148
More Information on Animation 149

IDL and Animation

IDL can help you visualize your data dynamically by using animation. An animation is just a series of still frames shown sequentially. In IDL, a series of frames can be represented by a 3-D array (for example, a 3-D array could hold forty, 300 pixel by 300 pixel images). This chapter shows you how to create an array of images and play them back as an animated sequence.

Note

To simplify obtaining useful results from the examples in this manual, create a bitmap buffer for your graphic windows and to use a maximum of 256 colors by entering the following command at the IDL command prompt:

```
DEVICE, RETAIN=2, DECOMPOSED=0
```

Animating a Series of Images

To create an animation that shows a series of images that represent an abnormal heartbeat, first read in the images to be displayed. The file holds 16 images of a human heart as 64 by 64 element arrays of bytes.

1. Enter the following commands at the IDL Command Line:

```
HEARTTEMPLATE=BINARY_TEMPLATE(FILEPATH('abnorm.dat',$
   SUBDIR=['examples', 'data']))
```

2. When the binary template dialog box appears, name the template “Animation” and then click **New Field**.
3. Enter “H” for the **Field Name**, be sure to specify that you have three dimensions and that the sizes are 64, 64 and 16.
4. Also select **Byte** in the **Type** field. Now click **OK** for both open dialogs.
5. Next, read the images into variable `HEART_BINARY`:

```
HEART_BINARY=READ_BINARY(FILEPATH('abnorm.dat',SUBDIR= $
   ['examples', 'data']), TEMPLATE=HEARTTEMPLATE)
```

6. Load an appropriate color table:

```
LOADCT, 3
```

7. Display the first “slice” of our 3-D array:

```
TV,HEART_BINARY.H[*, *, 0]
```

   The asterisks (*) in the first two element positions tell IDL to use all of the elements in those positions. Hence, the TV procedure displays a 64 by 64 byte image. The image is rather small.

8. Now resize each image in the array with bilinear interpolation by entering:

```
H=REBIN(HEART_BINARY.H,320,320,16)
```

9. Then display:

```
TV,H[*, *, 0]
```

   Each image in H is 5 times its previous size.

   Now a simple `FOR` statement can be used to “animate” the images. (A more robust and convenient animation routine, XINTERANIMATE, is described next.)

10. To animate, enter:

```
FOR I=0,15 DO TVSCL,H[*,*,i]
```

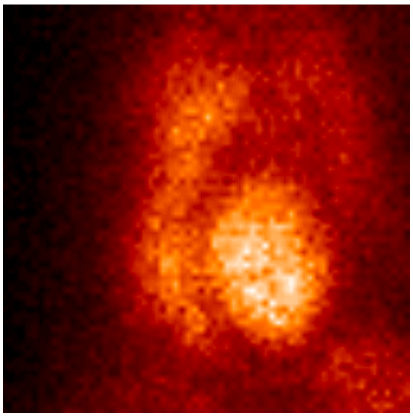


Figure 11-1: Representation of an abnormal heartbeat

IDL displays the 16 images in the array *H* sequentially. To repeat the animation, press the “up arrow” key to recall the command and press enter.

Note

If the IDLDE screen covers and existing IDL window, you may want to delete the current IDL window before recalling the `FOR` statement in order to clearly see the animation.

11. Dismiss the window:

```
WDELETE
```

Displaying an Animation as a Wire Mesh

The same series of images can be displayed as different types of animations. For example, each frame of the animation could be displayed as a SURFACE plot.

1. Create a new array to hold the heartbeat data:

   ```
   S=REBIN(HEART_BINARY.H,32,32,16)
   ```

   S now holds 32 byte by 32 byte versions of the heartbeat images. SURFACE plots are often more legible when made from a resized version of the dataset with fewer data points in it.

2. Display the first image in *S*, as a wire-mesh surface by entering:

   ```
   SURFACE,S[*,*,0]
   ```

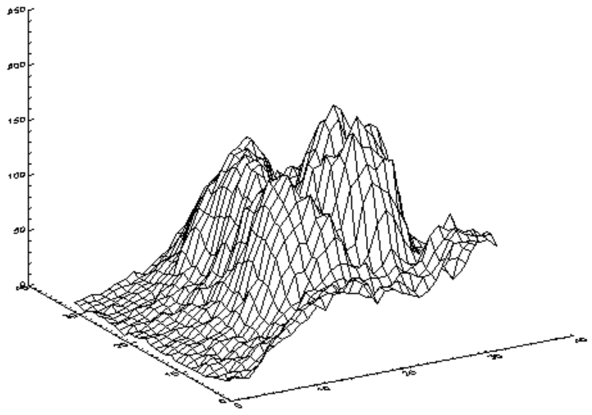


Figure 11-2: Surface visualization of heartbeat data

   Now create a whole series of SURFACE plots, one for each image in the original dataset.

3. To do this, first create a three-dimensional array to hold all of the images by entering:

   ```
   FRAMES=BYTARR(300,300,16)
   ```

   The variable *frames* will hold sixteen, 300 by 300 byte images.

4. Now create a 300 by 300 pixel window in which to display the images:

```
WINDOW,1,TITLE='IDL Animation',xsize=300,ysize=300
```

   A blank IDL Animation screen will appear.

   The next command will draw each frame of the animation. A SURFACE plot is drawn in the window and then the TVRD function is used to read the image from the plotting window into the *frames* array. The FOR loop is used to increment the array indices. The lines which follow are actually a single IDL command. The dollar sign ($) works as a continuation character in IDL and the ampersand (&) allows multiple commands in the same line.

5. Enter:

```
FOR I=0,15 DO BEGIN SURFACE,S[*,*,i],ZRANGE=[0,250]$
   & FRAMES[0,0,i]=TVRD()&END
```

   You should see a series of SURFACE plots being drawn in the animation window, as shown in below. The ZRANGE keyword is used to keep the “height” axis the same for each plot.

6. Now display the new images in series by entering:

```
FOR I=0,15 DO TV,FRAMES[*,*,i]
```

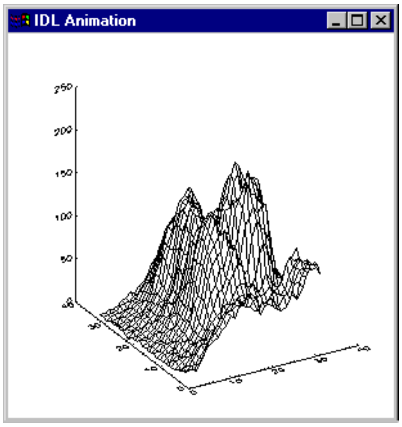


Figure 11-3: One of the SURFACE plots of the animation window

Note

Once again, if the IDLDE screen covers and existing IDL window, you may want to delete the current IDL window before recalling the `FOR` statement in order to clearly see the animation.

Animation with XINTERANIMATE

IDL includes a powerful, widget-based animation tool called XINTERANIMATE. Sometimes it is useful to view a single wire-mesh surface or shaded surface from a number of different angles. Let’s make a SURFACE plot from one of the S dataset frames and view it rotating through 360 degrees. by entering:

1. Save the first frame of the S dataset in the variable A to simplify the next set of commands:

   ```
   A=S[*,*,0]
   ```

2. Create a window in which to display your surface:

   ```
   WINDOW,0,XSIZE=300,YSIZE=300
   ```

3. Display A as a wire-mesh surface:

   ```
   SURFACE,A,XSTYLE=4,YSTYLE=4,ZSTYLE=4
   ```

   Setting the XSTYLE, YSTYLE, and ZSTYLE keywords equal to 4 turns axis drawing off. Usually, IDL automatically scales the axes of plots to best display all of the data points sent to the plotting routine. However, for this sequence of images, it is best if each SURFACE plot is drawn with the same size axes. The SCALE3 procedure can be used to control various aspects of the three-dimensional transformation used to display plots.

4. Force the X and Y axis ranges to run from 0 to 32 and the Z axis range to run from 0 to 250:

   ```
   SCALE3,XRANGE=[0,31],YRANGE=[0,31],ZRANGE=[0,250]
   ```

5. Set up the XINTERANIMATE routine to hold 40, 300 by 300 byte images:

   ```
   XINTERANIMATE,SET=[300,300,40],/SHOWLOAD
   ```

6. Return focus to the plot window for the SURFACE calls which follow.

   ```
   WSET, 0
   ```

7. Generate each frame of the animation and store it for the XINTERANIMATE routine. Once a 3-D transformation has been established, most IDL plotting routines can be made to use it by including the T3D keyword. The [XYZ]STYLE keywords are shortened to [XYZ]ST:

   ```
   FOR I=0,39 DO BEGIN SCALE3,AZ= -i * 9 & SURFACE,A, $
      /T3D,XSTYLE=4,YSTYLE=4,ZSTYLE=4 & XINTERANIMATE,$
      FRAME=I,WIN=0 & END
   ```

8. Play images back as an animation after all the images have been saved in the XINTERANIMATE routine:

   ```
   XINTERANIMATE
   ```

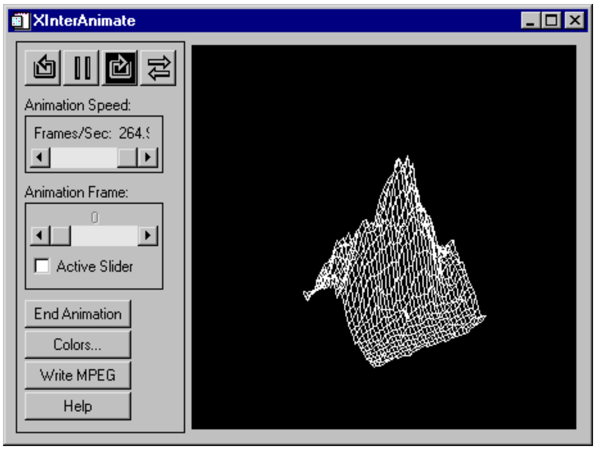


Figure 11-4: The XINTERANIMATE window

The XINTERANIMATE window should appear, as shown above. “Tape recorder” style controls can be used to play the animation forward, play it backward, or stop. Individual frames can also be selected by moving the “Animation Frame” slider. The “Options” menu controls the style and direction of image playback. Click on “End Animation” when you are ready to return to the IDL Command Line.

Cleaning Up the Animation Windows

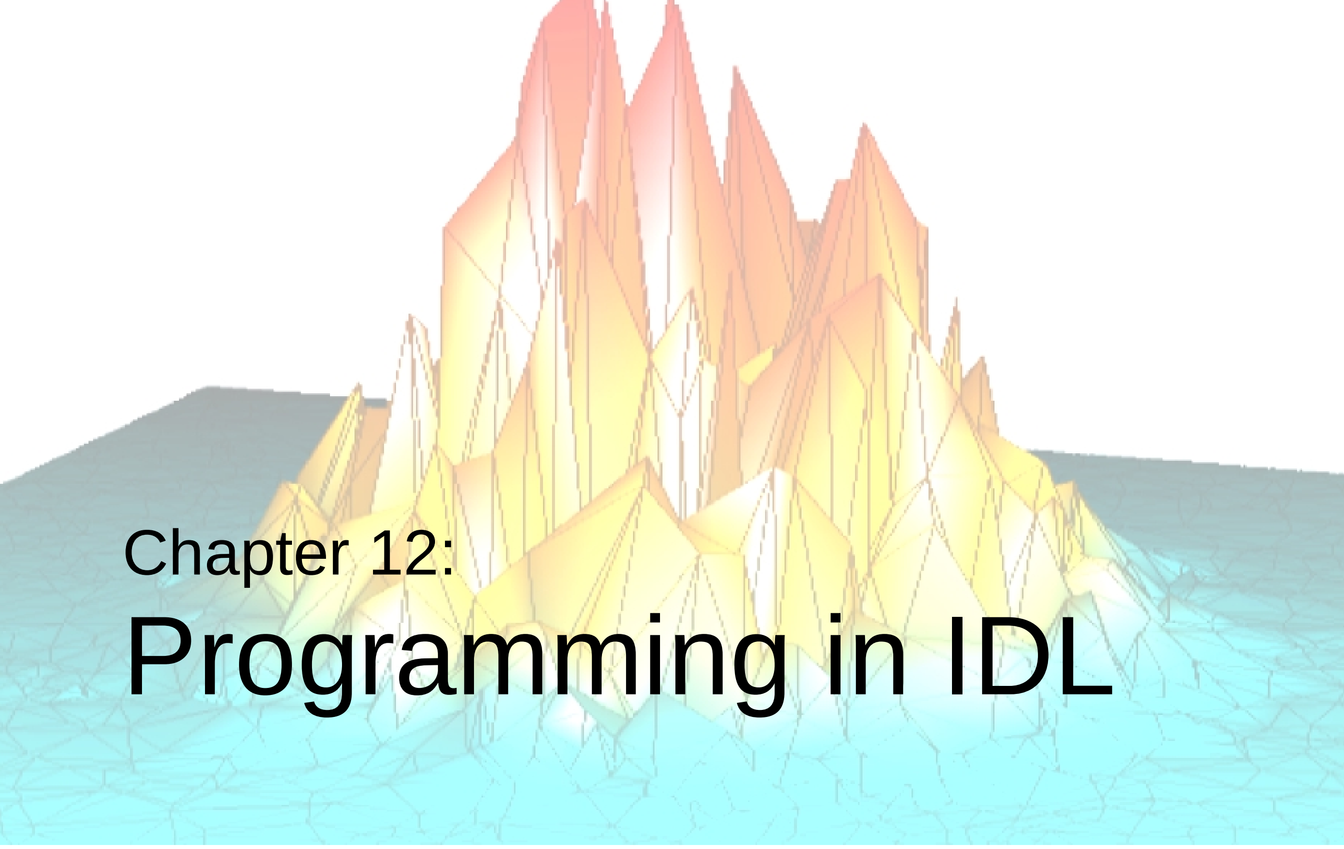
Before continuing with the rest of the tutorials, delete the two windows you used to create the animations. The WDELETE procedure is used to delete IDL windows.

1. Delete both window 0 and window 1 by entering:

```
WDELETE, 0
WDELETE, 1
```

More Information on Animation

With just a few IDL commands, you've created a number of different types of animation. For a list of other animation related commands, see the *IDL Reference Guide*.



Chapter 12: Programming in IDL

This chapter describes the following topics:

IDL and Programming 152
Programming Capabilities in the IDLDE . 153
Executing a Simple IDL Example Program 155
Debugging Tools in IDL 158
Using IDL Projects 160
More Information on IDL Programming . 162

IDL and Programming

IDL has a complete set of program control statements to write sophisticated programs and applications. These control statements are similar to, if not identical to, those found in other programming languages. This chapter demonstrates just some of IDL's basic programming capabilities.

Note

For best performance when using these examples, create a bitmap buffer for your graphic windows and to use a maximum of 256 colors by entering the following command at the IDL command prompt:

```
DEVICE, RETAIN=2, DECOMPOSED=0
```

Programming Capabilities in the IDLDE

IDL offers you the ability to program applications with ease. The term “IDL Application” is used very broadly; any program written in the IDL language is treated as an IDL application. IDL applications range from the very simple (a MAIN program entered at the IDL command line, for example) to the very complex (large programs with graphical user interfaces). Whether you are writing a small program to analyze a single data set or a large-scale application for commercial distribution, it is useful to understand the programming concepts used by the IDL language. IDL even allows you to call IDL from other programs written in other languages and call other programs from IDL.

Built-In Editor

The IDL Editor is a programmer’s-style editor—if you indent a line using the Tab key, the following lines will be indented as well. Use the Shift-Tab key to move left one tab stop. You can move the cursor position within an IDL Editor window using either the mouse or the keyboard. IDL Editor window key definitions are listed in *Using IDL*.

Chromacoded editor

On IDL for Windows, the IDL Editor supports chromacoding—different types of IDL statements appear in different colors. By default, the IDL Editor uses chroma-coding. The **Editor** tab from **Preferences** in the **File** menu displays the colors used for different words recognized by IDL. Change the Foreground color to change the color of the word itself. Highlight the word by specifying the Background color.

Types of IDL Programs

Main Program

A main program unit consists of a sequence of IDL statements that end in an END statement. Only one main program unit may exist in IDL at any time. All commands (except executive statements) that can be entered at the IDL Command Line can also be contained in an IDL program.

Procedure

A procedure is a self-contained sequence of IDL statements that performs a well-defined task. A procedure is identified by a procedure definition statement where the procedure name is the name of the IDL statement you are creating and the parameters are named variables that are used in the procedure.

Function

A function is a self-contained sequence of IDL statements that performs a well-defined task and returns a value to the calling program unit when it is executed. All functions return a function value which is given as a parameter in the RETURN statement used to exit the function.

Compound Statements

- BEGIN...END

Conditional Statements

- IF ... THEN ... ELSE
- CASE
- SWITCH

Loop Statements

- FOR...DO
- WHILE...DO
- REPEAT...UNTIL

Jump Statements

- BREAK
- CONTINUE
- GOTO

Note

For more information about these IDL statements, see Chapter 12, “Program Control” in *Building IDL Applications*.

Executing a Simple IDL Example Program

To show IDL's programming capabilities, enter a program here which will remove the bridges from the image of Manhattan Island in New York City using IDL's erosion and dilation power.

1. From the IDLDE, open a new IDL Editor window by selecting **File** → **New** → **Editor** (or for Macintosh, simply **File** → **New**).
2. Type (or copy) the following lines of code into the new Editor window to form a program:

```
pro remove_bridges
;
;Read an image of New York.
xsize = 768 ; pixels.
ysize = 512 ; pixels.
img = read_binary( $
    filepath('nyny.dat', subdir=['examples', 'data']), $
    data_dims=[xsize, ysize])
;
;Increase image's contrast.
img = bytscl(img)
;
;Create an image mask from thresholded image.
threshold_level = 70 ; determined empirically.
mask = img lt threshold_level
;
;Make a disk-shaped "structuring element."
disk_size = 7 ; determined empirically.
se = shift(dist(disk_size), disk_size / 2, disk_size / 2)
se = se le disk_size / 2
;
;Remove details in the mask's shape.
mask = dilate(erode(mask, se), se)
;
;Fuse gaps in the mask's shape.
mask = erode(dilate(mask, se), se)
;
;Remove all but the largest region in the mask.
label_img = label_region(mask)
labels = label_img[where(label_img ne 0)] ; Remove
background.
label = where(histogram(label_img) eq max(histogram(labels)))
mask = label_img eq label[0]
;
;Generate a new image consisting of local area minimums.
new_img = dilate(erode(img, se, /gray), se, /gray)
```

```
;
;Replace new image with original image, where not masked.
new_img[where(mask eq 0)] = img[where(mask eq 0)]
;
;View result, comparing the new image with the original.
print, 'Hit any key to end program.'
window, xsize=xsize, ysize=ysize
flick, img, new_img
wdelete
end
```

Note

Semicolons (;) in IDL code are indicators of the beginning of comment lines, which explain what the actual code lines are doing and/or to help you understand your code (while being ignored by IDL itself).

Note

The dollar sign ($) at the end of the first line is the IDL continuation character. It allows you to enter long IDL commands as multiple lines.

Note

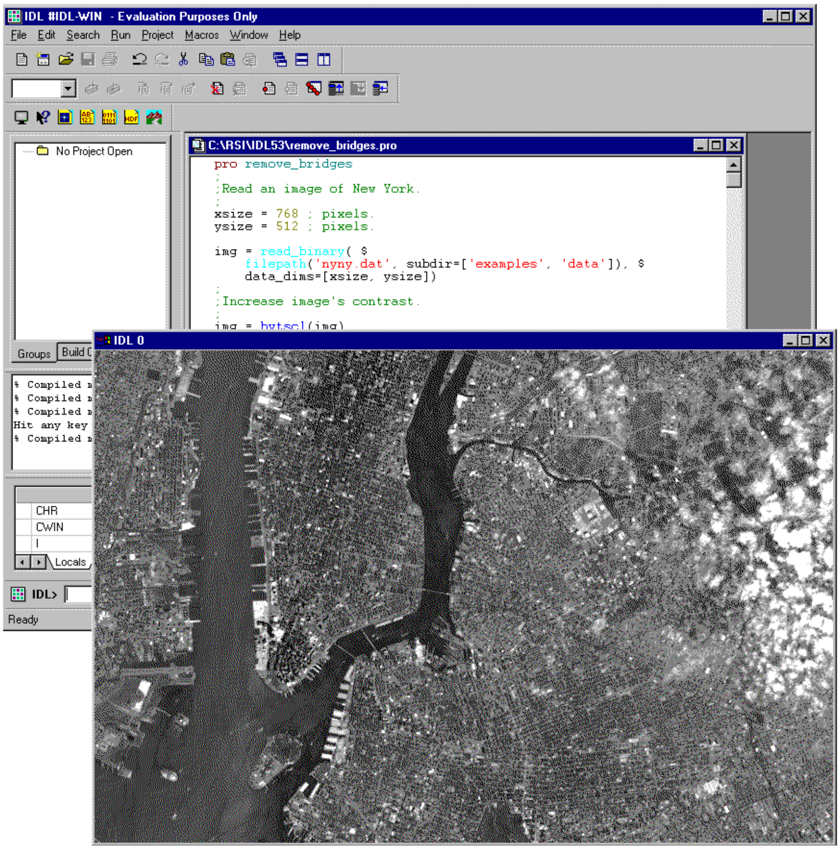
For the current settings for the various colors represented in the chromacoded editor go to **File** → **Preferences** and then select the **Editor** tab (on Macintosh, select **File** → **Preferences** → **Syntax Coloring**.

Saving, Compiling and Running your Program

Now to see the program in action, IDL requires a few more simple steps.

1. Save the file as remove_bridges.pro by selecting File → Save As and then entering “remove_bridges.pro”.
2. Compile the program by selecting **Run** → **Compile remove_bridges.pro** (or on Macintosh, simply **Run** → **Compile**).

3. Run the program by selecting **Run** → **Run remove_bridges.pro** (or for Macintosh, simply **Run** → **Run**). You'll see the following:


IDL #IDL-WIN - Evaluation Purposes Only
File Edit Search Run Project Macros Window Help
No Project Open
Groups Build C
C:\RSI\IDL53\remove_bridges.pro
IDL 0
CHR
CWIN
I
Locals
IDL>
Ready

```
pro remove_bridges
;
;Read an image of New York.
;
xsize = 768 ; pixels.
ysize = 512 ; pixels.

img = read_binary( $
    filepath('nyny.dat', subdir=['examples', 'data']), $
    data_dims=[xsize, ysize])
;
;Increase image's contrast.
;
img = bytscl(img)
```

```
% Compiled m
% Compiled m
% Compiled m
Hit any key
% Compiled m
```

Figure 12-1: Running your IDL Program

Note

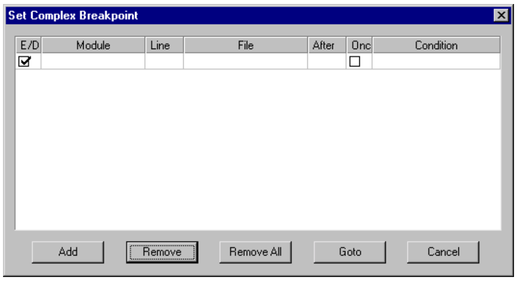
If your program encounters an error in running be sure to double check your code for typographical errors.

Debugging Tools in IDL

Many features are included to help you debug your IDL programs. These features are discussed in the following sections:

Breakpoints

This allows you to set points in your program to stop the execution of the program. You can then check the value of variables at those points to see what is happening in your program and then continue execution of the program.



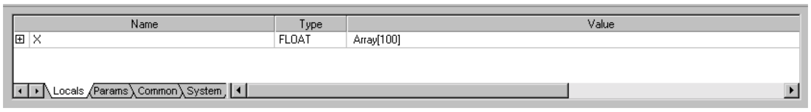
E/D	Module	Line	File	After	Onc	Condition
☑					☐	

Figure 12-2: Complex Breakpoint Dialog

Variable Watch window

This window displays current variable values after IDL has completed execution. If the calling context changes during execution — as when stepping into a procedure or function — the variable table is replaced with a table appropriate to the new context.

While IDL is at the main program level, the Watch window remains active and displays any variables created.



Name	Type	Value
⊞ X	FLOAT	Array[100]

Figure 12-3: Variable Watch Window

The IDL Code Profiler

The IDL Code Profiler helps you analyze the performance of your applications. You can easily monitor the calling frequency and execution time for procedures and functions. The Profiler can be used with programs entered from the command line as well as programs run from within a file.

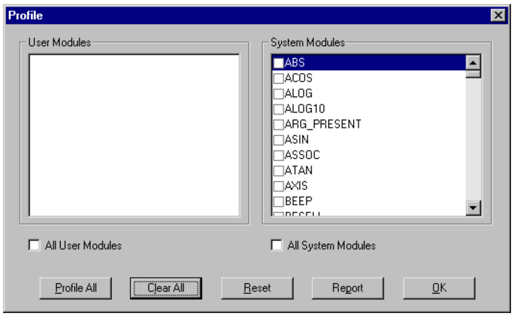
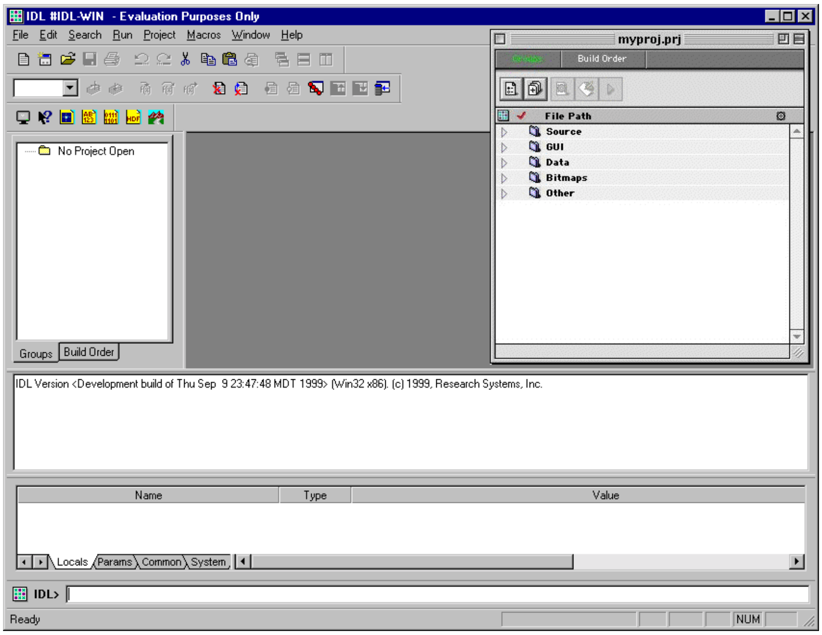


Figure 12-4: Profile Dialog

Using IDL Projects

IDL Projects allow you to easily develop applications in IDL. You can manage, compile, run, and create distributions of all the files you will need to develop your IDL application. All of your application files can be organized so that they are easier to access and easier to export to other developers, colleagues, or users.



```
IDL Version <Development build of Thu Sep  9 23:47:48 MDT 1999> (Win32 x86). (c) 1999, Research Systems, Inc.
```

Figure 12-5: Projects Window for Macintosh (upper right) and Windows (upper left)

Access to all Files in Your Application

IDL Projects have an easy to use interface for grouping:

- IDL source code files (.pro)
- GUI files (.prc) created with the IDL GUIBuilder
- Data files (ASCII text or binary)
- Image files (.tif, .jpg, .bmp, etc.)

- Other files (help files, .sav files, etc.)

After you add all of your files to your project, you can simply double click on `.pro` files to open them in the IDL editor or `.prc` files to open them in the IDL GUIBuilder.

Working with Files in Your Project

IDL projects makes it easy to add, remove, move, edit, compile, and test files in your project.

All of your workspace information is saved as well. If you save and exit your project with the files you are working on open, when you open your project, those same files will be opened automatically for you.

IDL projects also store breakpoint information. There is no need to reset breakpoints every time you open the project.

Compiling and Running Your Application

Compiling and running applications is fast and easy. You can compile all of your source files or just the files that you have modified and then run your application through the Projects menu. You can customize how your application is compiled and run by specifying options for your project.

Building Distributions

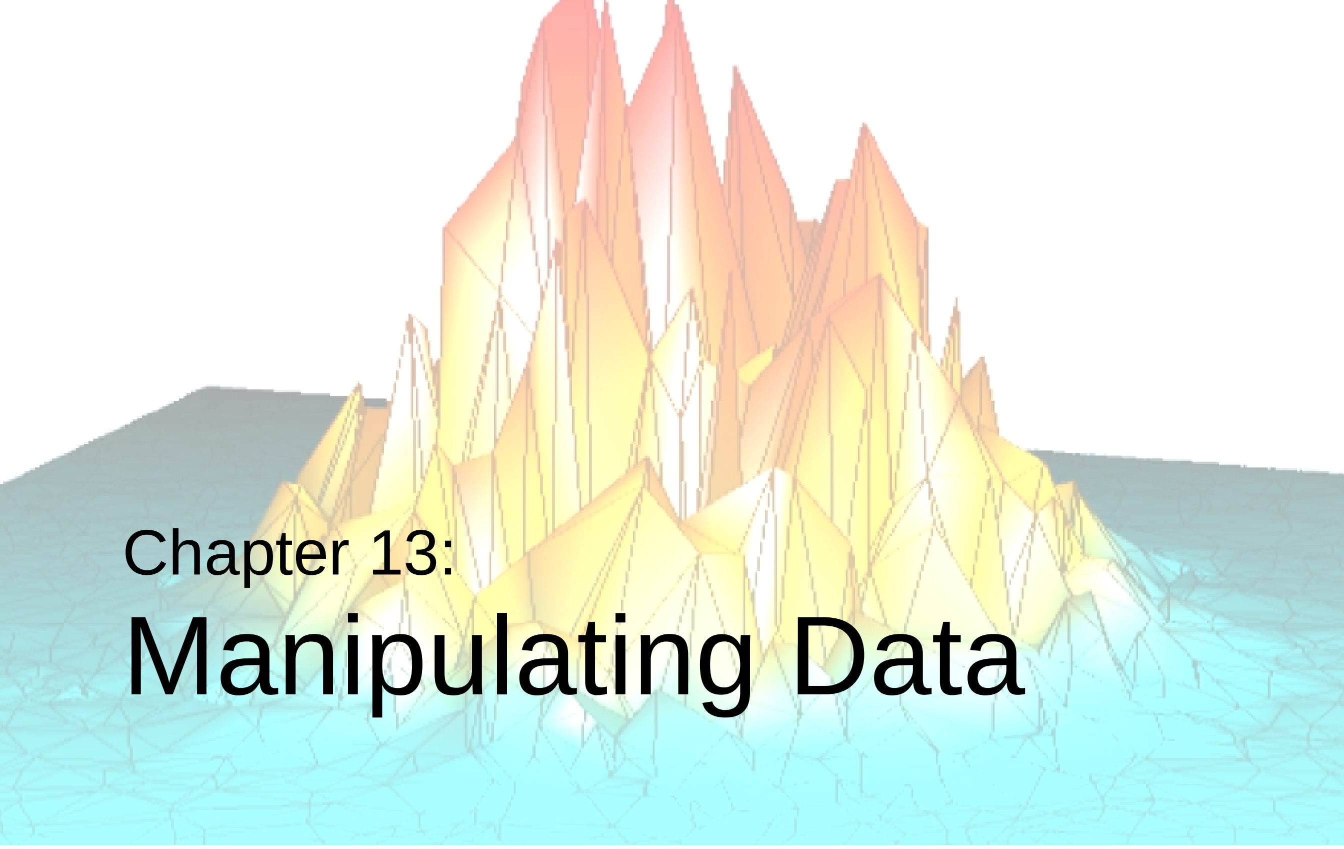
Once you have completed your application, you can quickly and easily create a distribution. If you have purchased the IDL Developer’s Kit, your application is automatically licensed for distribution.

Exporting Your Applications

You can easily move your application to another platform or distribute your source code to colleagues by exporting your project. All your source code, GUI files, data files, and image files are copied to a directory you specify. You also have the option of creating an IDL Run Time distribution with your application.

More Information on IDL Programming

For more information on programming in IDL, see the *Building IDL Applications* manual. Also see the documentation for specific routines in the *IDL Reference Guide*.



Chapter 13: Manipulating Data

This chapter describes the following topics:

IDL and Manipulating Data 164
IDL Array Routines 165
Array Processing Capabilities 168
More Information on Manipulating Data . 171

IDL and Manipulating Data

IDL has been specifically designed to process arrays easily and naturally. You can get excellent performance in your applications by using the built-in array processing routines instead of other methods like FOR loops. This chapter will show how easy it is to manipulate your data using IDL's capabilities.

IDL Array Routines

The following tables describe some of the array processing procedures and functions that are included in IDL.

Array Creation Routines

Array Name	Array Function
BINDGEN	Return a byte array with each element set to its subscript
BYTARR	Create a byte vector or array
CINDGEN	Returns complex array with each element set to its subscript
COMPLEXARR	Returns complex, single-precision, floating-point vector or array
DBLARR	Return a double-precision array
DCINDGEN	Return a double-precision, complex array with each element set to its subscript
DCOMPLEXARR	Return a double-precision, complex vector or array
DINDGEN	Return a double-precision array with each element set to its subscript
FINDGEN	Return a floating-point array with each element set to its subscript
FLTARR	Return a floating-point vector or array
IDENTITY	Return an identity array
INDGEN	Return an integer array with each element set to its subscript
INTARR	Return an integer vector or array
LINDGEN	Return a longword integer array with each element set to its subscript
LONARR	Return a longword integer vector or array
MAKE_ARRAY	General purpose array creation

Table 13-1: Array Creation Routines

Array Name	Array Function
OBJARR	Create an array of object references
PTRARR	Create an array of pointers
REPLICATE	Form array of given dimensions filled with a value
SINDGEN	Return a string array with each element set to its subscript
STRARR	Return a string vector or array

Table 13-1: Array Creation Routines

Array Manipulation Routines

Array Name	Array Function
INVERT	Compute inverse of a square array
REFORM	Change array dimensions without changing contents
REVERSE	Reverse vectors or arrays
ROT	Rotate array by any amount
ROTATE	Rotate array by multiples of 90 degrees and/or transpose
SHIFT	Shift array elements
SORT	Sort array contents and return vector of indices
TRANSPOSE	Transpose array

Table 13-2: Array Manipulation Routines

Array and Image Processing Routines

Array Name	Array Function
CONGRID	Resample image to any dimensions
INVERT	Compute inverse of a square array
MAX	Return the maximum element of an array

Table 13-3: Array and Image Processing Routines

Array Name	Array Function
MEDIAN	Median function and filter
MIN	Return the minimum element of an array
REBIN	Resample array by integer multiples
REFORM	Change array dimensions without changing contents
REVERSE	Reverse vectors or arrays
ROT	Rotate array by any amount
ROTATE	Rotate array by multiples of 90 degrees and/or transpose
SHIFT	Shift array elements
SIZE	Return array size and type information
SORT	Sort array contents and return vector of indices
TOTAL	Sum array elements
TRANSPOSE	Transpose array
WHERE	Return subscripts of non-zero array elements

Table 13-3: Array and Image Processing Routines

Array Processing Capabilities

Programs with array expressions run faster than programs with scalars, loops, and IF statements. Some examples of slow and fast ways to achieve the same results follow.

Example— Avoiding IF Statements by Summing Elements

The first example adds all positive elements of array B to array A.

- Using a loop will be slow:

```
FOR I=0,(N-1)DO IF B[I]GT 0 THEN A[I]=A[I] + B[I]
```

- Fast way: Mask out negative elements using array operations.

```
A=A + (B GT 0) * B
```

- Faster way: Add B > 0

```
A=A + (B > 0)
```

When an IF statement appears in the middle of a loop with each element of an array in the conditional, the loop can often be eliminated by using logical array expressions.

Example— Avoiding IF Statements by Using Array Operators and the WHERE Function

In the example below, each element of C is set to the square-root of A if A[I] is positive; otherwise, C[I] is set to minus the square-root of the absolute value of A[I].

- Using an IF statement is slow.

```
FOR I=0,(N-1) DO IF A[I] LE 0 THEN C[I]=-SQRT(-A[I]) ELSE
   C[I]=SQRT(A[I])
```

- Fast way

```
C = ((A GT 0) * 2 - 1 ) * SQRT(ABS(A))
```

The expression `(A GT 0)` has the value 1 if A[I] is positive and has the value 0 if A[I] is not. `(A GT 0)* 2 - 1` is equal to +1 if A[I] is positive or -1 if A[I] is negative, accomplishing the desired result without resorting to loops or IF statements.

Another method is to use the WHERE function to determine the subscripts of the negative elements of A and negate the corresponding elements of the result.

- Get subscripts of negative elements.

```
NEGS=WHERE(A LT 0)
```

- Take root of absolute value.

```
C = SQRT(ABS(A))
```

- Negate elements in C corresponding to negative elements in A.

```
C[negs] = -C[negs]
```

Example— Using Vector and Array Operations

Whenever possible, vector and array data should always be processed with IDL array operations instead of scalar operations in a loop. For example, consider the problem of flipping a 512×512 image. This problem arises because approximately half the available image display devices consider the origin to be the lower-left corner of the screen, while the other half recognize it as the upper-left corner.

The following example is for demonstration only. The IDL system variable !ORDER should be used to control the origin of image devices. The ORDER keyword to the TV procedure serves the same purpose.

A programmer without experience in using IDL might be tempted to write the following nested loop structure to solve this problem:

```
FOR I = 0, 511 DO FOR J = 0, 255 DO BEGIN
```

- Temporarily save pixel:

```
TEMP=IMAGE[I, J]
```

- Exchange pixel in same column from corresponding row at bottom.

```
image[I, J] = image[I, 511 - J]
image[I, 511-J] = temp
ENDFOR
```

A more efficient approach to this problem capitalizes on IDL's ability to process arrays as a single entity.

- Enter at the IDL Command Line:

```
FOR J = 0, 255 DO BEGIN
```

- Temporarily save current row.

```
temp = image[*, J]
```

- Exchange row with corresponding row at bottom.

```
image[*, J] = image[*, 511-J]
image[*, 511-J] = temp
ENDFOR
```

At the cost of using twice as much memory, processing can be simplified even further by using the following statements:

- Get a second array to hold inverted copy.

```
image2 = BYTARR(512, 512)
```

- Copy the rows from the bottom up.

```
FOR J = 0, 511 DO image2[*, J] = image[*, 511-J]
```

- Even more efficient is the single line:

```
image2 = image[*, 511 - INDGEN(512)]
```

that reverses the array using subscript ranges and array-valued subscripts.

- Finally, using the built-in ROTATE function is quickest of all:

```
image = ROTATE(image, 7)
```

  This works because inverting the image is equivalent to transposing it and rotating it 270 degrees clockwise.

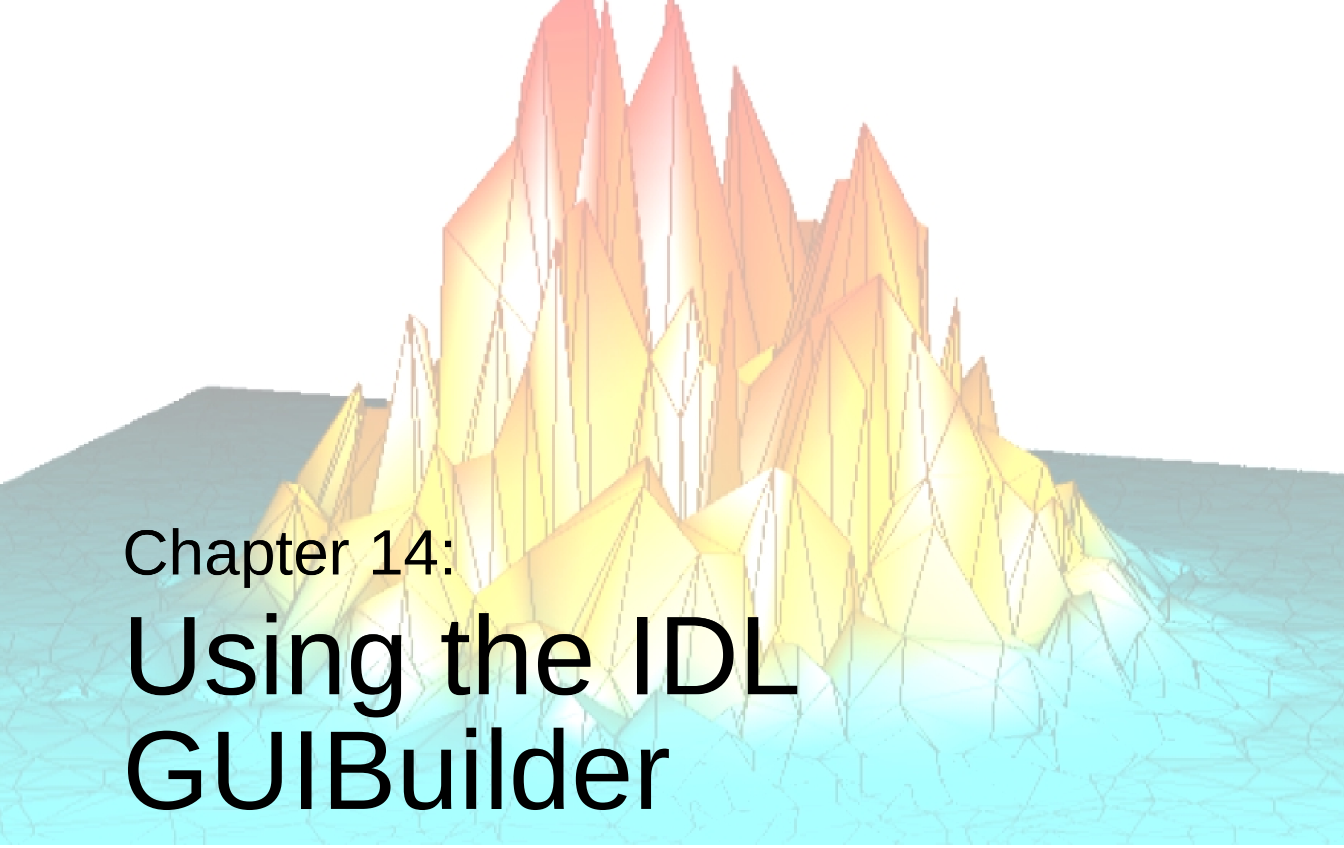
Note

Another way to invert the image is to enter:

```
image = REVERSE(image, 2)
```

More Information on Manipulating Data

IDL has many more array processing capabilities than the ones shown in this chapter. To take advantage of all of IDL's powerful capabilities in manipulating data, look for more information in *Building IDL Applications*.



Chapter 14: Using the IDL GUIBuilder

This chapter describes the following topics:

What is the IDL GUIBuilder? 174
IDL GUIBuilder Tools 176
Creating an Example Application 177
Widget Types 187
More Information on the IDL GUIBuilder 189

What is the IDL GUIBuilder?

The IDL GUIBuilder is part of the IDLDE for Windows. The IDL GUIBuilder supplies you with a way to interactively create user interfaces and then generate the IDL source code that defines that interface and contains the event-handling routine place holders.

Note

The IDL GUIBuilder is supported on Windows only. However, the code it generates is portable to other platforms and will run on the same version of IDL or higher.

The IDL GUIBuilder has several tools that simplify application development. These tools allow you to create the widgets that make up user interfaces, define the behavior of those widgets, define menus, and create and edit color bitmaps for use in buttons.

Note

When using code generated by the IDL GUIBuilder on other non-Windows platforms, more consistent results are obtained by using a row or column layout for your bases instead of a bulletin board layout. By using a row or column layout, problems caused by differences in the default spacing and decorations (for example, beveling) of widgets on each platform can be avoided

Using the IDL GUIBuilder

These are the basic steps you will follow when building an application interface using the IDL GUIBuilder:

1. Interactively design and create a user interface using the components, or *widgets*, supplied in the IDL GUIBuilder. Widgets are simple graphical objects supported by IDL, such as sliders or buttons.
2. Set attribute properties for each widget. The attributes control the display, initial state, and behavior of the widget.
3. Set event properties for each widget. Each widget has a set of events to which it can respond. When you design and create an application, it is up to you to decide if and how a widget will respond to the events it can generate. The first step to having a widget respond to an event is to supply an event procedure name for that event.
4. Save the interface design to an IDL resource file, *.prc file, and generate the portable IDL source code files. There are two types of generated IDL source

code: widget definition code (`*.pro` files) and event-handling code (`*_eventcb.pro` files).

5. Modify the generated `*_eventcb.pro` event-handling code file using the IDLDE, then compile and run the `*.pro` code. This code can run on any IDL-supported platform.

The `*_eventcb.pro` file contains place holders for all of the event procedures you defined for the widgets, and you complete the file by filling in the necessary event callback routines for each procedure.

Warning

Once you have generated the widget definition code (`*.pro` files), you should not modify this file manually. If you decide to change your interface definition, you will need to regenerate the interface code, and will therefore overwrite that `*.pro` file. The event handling code will not be overwritten but will instead be appended.

Chapter 24, “Widgets” in *Building IDL Applications* contains complete information about IDL widgets, and it describes how to create user interfaces programmatically (without the IDL GUIBuilder).

IDL GUIBuilder Tools

You will use the following tools to design and construct a graphical interface using the IDL GUIBuilder:

- The IDL GUIBuilder Toolbar, which you use to create the widgets that make up your interface.
- Widget Properties dialog, which you use to set widget attributes and event properties.
- Widget Browser, which you can use to see the widget hierarchy and to modify certain aspects of the widgets in your application.
- The Menu Editor, which you use to define menus to top-level bases and buttons.
- The Bitmap Editor, which you use to create or modify bitmap images to be displayed on button widgets.
- The IDLDE to modify, compile, and run the generated code.

Using the IDL GUIBuilder Toolbar

The IDL GUIBuilder has its own toolbar in the IDE, which you use to create the widgets for your user interface.

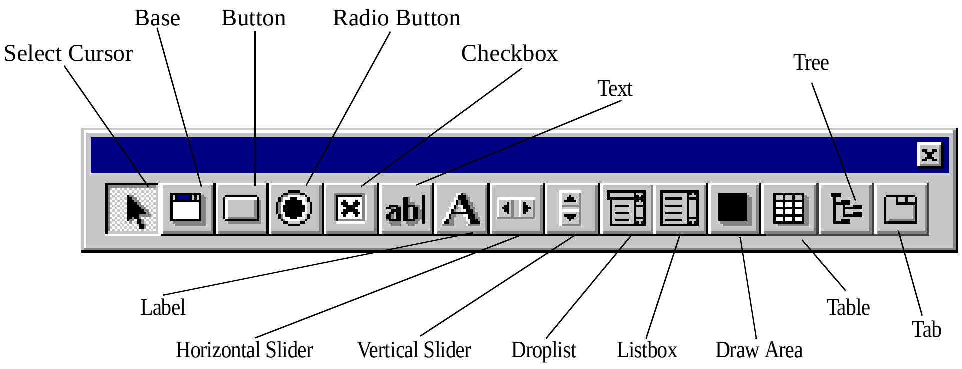


Figure 14-1: IDL GUIBuilder Toolbar

Creating an Example Application

This simple example application contains a menu and a draw widget. When complete, the running application allows the user to open and display a graphics file in PNG format, change the color table for the image display, and perform a smooth operation on the displayed image.

Now let’s create a widget, set widget properties, and write IDL code to handle events:

Defining Menus for the Top-Level Base

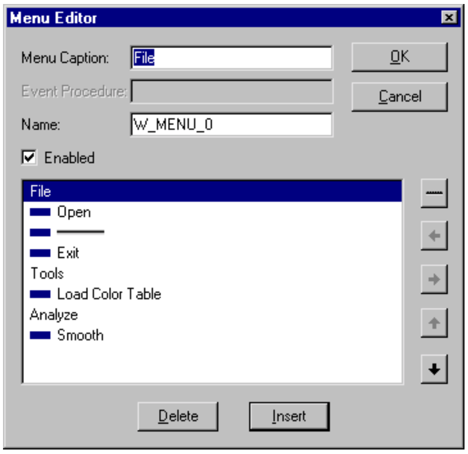
To define the menu, follow these steps:

1. Open a new IDL GUIBuilder window by selecting **File** → **New** → **GUI** from the IDLDE menu, or click the “New GUI” button on the IDLDE toolbar.
2. Drag out the window so that the top-level base to a reasonable size for displaying an image.

   To view the property values, right-click on the base, and choose Properties from menu. In the **Properties** dialog, scroll down to view the X Size and Y Size property values.
3. Right-click on the top-level base in the IDL GUIBuilder window, then choose Edit Menu. This action opens the Menu Editor.
4. In the Editor’s **Menu Caption** field, enter “File” and click Insert. Clicking Insert sets the entered value and adds a new line after the currently selected line, and the new line becomes the selected line.
5. To define the File menu items, do the following:
   - With the new line selected, click on the right arrow in the Menu Editor, which indents the line and makes it a menu item.
   - Click in the **Menu Caption** field and enter “Open”.
   - Click in the **Event Procedure** field and enter “OpenFile”. The OpenFile routine will be called when the user selects this menu.
   - To create a separator after the Open menu, click the line button at the right side of the dialog (above the arrow buttons).
   - To set the values and move to a new line, click Insert.
   - In the **Menu Caption** field, enter “Exit”.
   - In the **Event Procedure** field, enter “OnExit”.

- To set the values and move to a new line, click Insert.

6. To define the **Tools** menu and its one item, do the following:
   - With the new line selected, click the left arrow to make the line a top-level menu.
   - In the **Menu Caption** field, enter “Tools”, then click Insert.
   - Click the right arrow to make the new line a menu item.
   - In the **Menu Caption** field, enter “Load Color Table”.
   - In the **Event Procedure** field, enter “OnColor”.
   - To set the values and move to a new line, click Insert.
7. To define the **Analyze** menu and its one menu item, do the following:
   - With the new line selected, click the left arrow to make the line a top-level menu.
   - In the **Menu Caption** field, type “Analyze”, then press Enter.
   - Click the right arrow to make the new line a menu item.
   - In the **Menu Caption** field, enter “Smooth”.
   - In the **Event Procedure** field, enter “DoSmooth”.


Menu Editor
Menu Caption: File
Event Procedure:
Name: W_MENU_0
☑ Enabled

```
File
    Open
    ————
    Exit
Tools
    Load Color Table
Analyze
    Smooth
```

Figure 14-2: Menu Editor Dialog with Example Menus

8. Save your menu definitions by clicking OK in the Menu Editor.
9. At this time you can click on the menus to test them.
10. From the IDLDE **File** menu, choose **Save**, which opens the “Save As” dialog.
11. In the “Save As” dialog, select a location, enter “example.prc” in the File name field, and click Save. This action writes the portable resource code to the specified file.

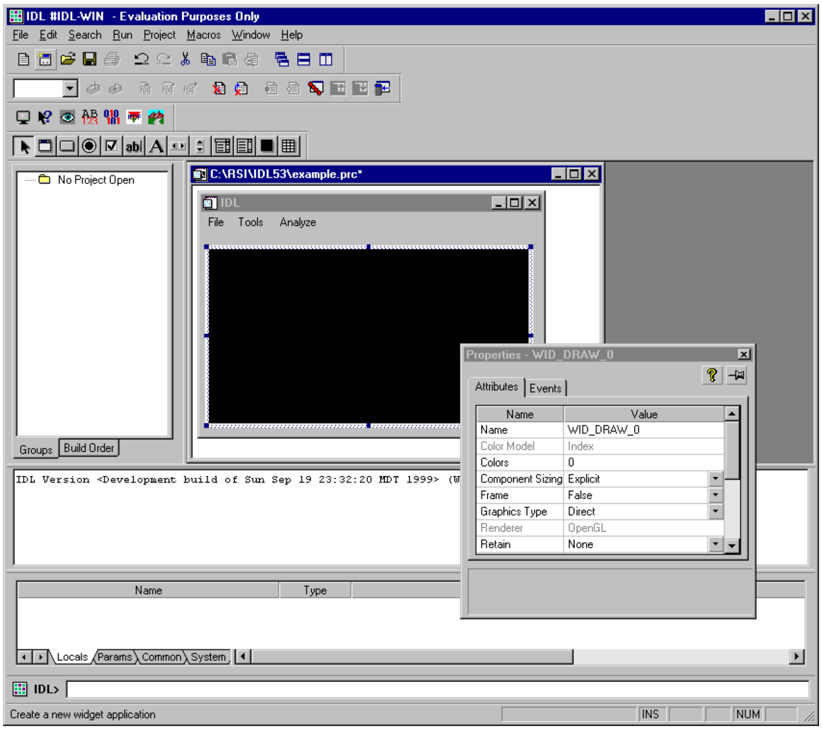
To create a draw area that will display PNG image files, follow these steps:

1. Click on the **Draw Widget** tool button (the dark square icon), then drag out an area that fills the top-level base display area. Leave a small margin around the edge of the draw area when you drag it out.
2. Right click on the draw area, and choose **Properties**. This action opens the **Properties** dialog for the draw area; the draw widget properties are displayed in the dialog.
3. In the **Properties** dialog, click the push pin button (in the top right corner of the dialog box) so the dialog will stay open and on top.

Note

This **Properties** dialog floats and is resizeable.

4. In the **Properties** dialog, change the draw widget **Name** attribute value to “Draw”.


IDL #IDL-WIN - Evaluation Purposes Only
File Edit Search Run Project Macros Window Help
No Project Open
Groups Build Order
C:\RSI\IDL53\example.prc*
IDL
File Tools Analyze
Properties - WID_DRAW_0
Attributes Events
Name Type
Locals Params Common System
IDL>
Create a new widget application
INS NUM

```
IDL Version <Development build of Sun Sep 19 23:32:20 MDT 1999> (W
```

Name	Value
Name	WID_DRAW_0
Color Model	Index
Colors	0
Component Sizing	Explicit
Frame	False
Graphics Type	Direct
Renderer	OpenGL
Retain	None

Figure 14-3: Changing the Name attribute to "Draw"

Later, you will write code to handle the display of the image in this draw area widget. Renaming the widget now will make it easier to write the code later; the "Draw" name is easy to remember and to type.

Note

The Name property must be unique to the widget hierarchy.

5. In the IDL GUIBuilder window, click on the top-level base widget to select it. When you do so, the **Properties** dialog will update and display the attributes for this base widget.
6. In the **Properties** dialog, locate the **Component Sizing** property, and select **Default** from the droplist values. This action sizes the base to the draw widget size you created.

When you first dragged out the size of the base, the **Component Sizing** property changed from **Default** to **Explicit**—you explicitly sized the widget. Now that the base widget contains items, you can return it to **Default** sizing, and IDL will handle the sizing of this top-level base.

7. From the **File** menu, choose **Save**.

Running the Application in Test Mode

You can run the application in test mode, which allows you to test the display of widgets and menus.

To run your application in test mode:

- From the **Run** menu, choose **Test GUI**.

  This action displays the interface as it will look when it runs.

To exit test mode:

- Press the **Esc** key or Click the close X in the upper-right corner of the test application window.

Generating the IDL Code

To generate the code for the example application, follow these steps:

1. From the **File** menu, choose **Generate .pro**. This action opens the “Save As” dialog.
2. In the “Save As” dialog, find the location where you want the files saved, enter “example.pro” in the File name field, and click Save.

This action generates an `example.pro` widget definition file and an `example_eventcb.pro` event-handling file.

The `example.pro` file contains the widget definition code, and you should never modify this file. If you decide later to change your interface, you will need to regenerate this interface code, and thus overwrite the widget code file.

The `example_eventcb.pro` contains place holders for all the event procedures you defined in the IDL GUIBuilder Menu Editor and Properties dialog. You must complete these event procedures by filling in event callback routines. This file will only be appended to when new event handlers are added so changes made will not be lost.

Note

You should modify *only* the generated event-handling file (`*_eventcb.pro`); you should never modify the generated interface code (the `*.pro` file).

Handling the Open File Event

You can now modify the generated `example_eventcb.pro` file to handle the events for the application. First, you will modify the OpenFile routine.

When the user selects **Open** from the File menu of the example application, the appropriate event structure is sent, and the OpenFile routine handles the event. For this application, the **Open** menu item will launch an **Open** dialog to allow the user to choose a PNG file, and then the routine will check the selected file's type, read the image, and display it in the draw area.

To open the file and add the code to handle the OpenFile event, follow these steps:

1. From the **File** menu in the IDLDE, choose **Open**, which launches the **Open** dialog.
2. In the **Open** dialog, locate and select the `example_eventcb.pro` file, and click **Open**. This file contains the event handling routine place holders, which you will now complete.
3. In the `example_eventcb.pro` file, locate the OpenFile procedure, which looks like this:

```
pro OpenFile, Event

end
```

Tip

To easily find the OpenFile routine, select OpenFile from the Functions/Procedures drop-down list on the IDLDE toolbar.

4. Add the following code between the PRO and END statements to handle the event:

```
; If there is a file, draw it to the draw widget.
sFile = DIALOG_PICKFILE(FILTER='*.png')
IF(sFile NE "")THEN BEGIN
   ; Find the draw widget, which is named Draw.
   wDraw = WIDGET_INFO(Event.top, FIND_BY_UNAME='Draw');
   ; Make sure something was found.
   IF(wDraw GT 0)THEN BEGIN
      ; Make the draw widget the current, active window.
```

```
            WIDGET_CONTROL, wDraw, GET_VALUE=idDraw
            WSET,idDraw
            ; Read in the image.
            im = READ_PNG(sFile, r, g, b)
            ; If TrueColor image, quantize image to pseudo-color:
            IF (SIZE(im, /N_DIM) EQ 3) THEN $
               im = COLOR_QUAN(im, 1, r, g, b)
            ; Size the image to fill the draw area.
            im = CONGRID(im, !D.X_SIZE, !D.Y_SIZE)
            ; Handle TrueColor displays:
            DEVICE, DECOMPOSED=0
            ; Load color table, if one exists:
            IF (N_ELEMENTS(r) GT 0) THEN TVLCT, r, g, b
            ; Display the image.
            TV, im
            ; Save the image in the uvalue of the top-level base.
            WIDGET_CONTROL, Event.top, SET_UVALUE=im, /NO_COPY
         ENDIF
      ENDIF
```

Note

In the added code, you used the FIND_BY_UNAME keyword to find the draw widget using its name property. In this example, the widget name, “Draw”, is the one you gave the widget in the IDL GUIBuilder Properties dialog. The widget name is case-sensitive.

Now re-save `example.pro` to be sure that the changes are retained.

Handling the Exit Event

To add the code that causes the example application to close when the user chooses **Exit** from the **File** menu, follow these steps:

1. Locate the OnExit procedure place holder, which looks like this:

```
pro OnExit, Event

end
```

2. Add the following statement between the PRO and END statements to handle the destruction of the application:

```
WIDGET_CONTROL, Event.top, /DESTROY
```

Handling the Load Color Table Event

To add the code that causes the example application to open the IDL color table dialog when the user chooses **Load Color Table** from the **Tools** menu, follow these steps:

1. Locate the OnColor routine place holder, which looks like this:

```
pro OnColor, Event

end
```

2. Add the following code between the PRO and END statements:

```
XLOADCT, /BLOCK
; Find the draw widget, which is named Draw:
wDraw = WIDGET_INFO(Event.top, FIND_BY_UNAME='Draw')
IF(wDraw GT 0) THEN BEGIN
   ; Make the draw widget the current, active window:
   WIDGET_CONTROL, wDraw, GET_VALUE=idDraw
   WSET, idDraw
   WIDGET_CONTROL,Event.top, GET_UVALUE=im, /NO_COPY
   ; Make sure the image exists:
   IF (N_ELEMENTS(im) NE 0) THEN BEGIN
   ; Display the image:
      TV, im
      ; Save the image in the uvalue of the top-level base:
      WIDGET_CONTROL, Event.top, SET_UVALUE=im, /NO_COPY
   ENDIF
ENDIF
```

This procedure opens a dialog from which the user can select from a set of predefined color tables. When the user clicks the name of a color table, it is loaded and the displayed image changes appropriately.

Note

The XLOADCT color table dialog affects only 8-bit display devices.

Handling the Smooth Event

When the user selects **Smooth** from the **Analyze** menu, a smooth operation is performed on the displayed image. The smooth operation displays a smoothed image with a boxcar average of the specified width, which in the example code is 5.

To add the callback routines to handle the smooth operation, follow these steps:

1. Locate the DoSmooth routine place holder, which looks like this:

```
pro DoSmooth, Event

end
```

2. Add the following code between the PRO and END statements to handle the smooth operation:

```
; Get the image stored in the uvalue of the top-level-base.
WIDGET_CONTROL, Event.top, GET_UVALUE=image, /NO_COPY
; Make sure the image exists.
IF(N_ELEMENTS(image) GT 0)THEN BEGIN
   ; Smooth the image.
   image = SMOOTH(image, 5)
   ; Display the smoothed image.
   TV, image
   ; Place the new image in the uvalue of the button widget.
   WIDGET_CONTROL, Event.top, SET_UVALUE=image, /NO_COPY
     ENDIF
```

3. From the **File** menu, choose **Save**, which saves all your changes to the `example_eventcb.pro` file.

Compiling and Running the Example Application

To compile and run your example application, follow these steps:

1. Type `example` at the IDL> command prompt. This compiles and runs the example application, opening the GUI interface that has been created.
2. Now open a PNG image to try out the new application. From the **File** menu choose **Open**, locate a PNGfile, and click “Open”.
3. You will now see the image opened in your GUI window. To manipulate the color schemes, click **Tools**/**Load Color Table** The following figure shows the example application and the IDL color table dialog. You can also perform the smooth procedure on the image.

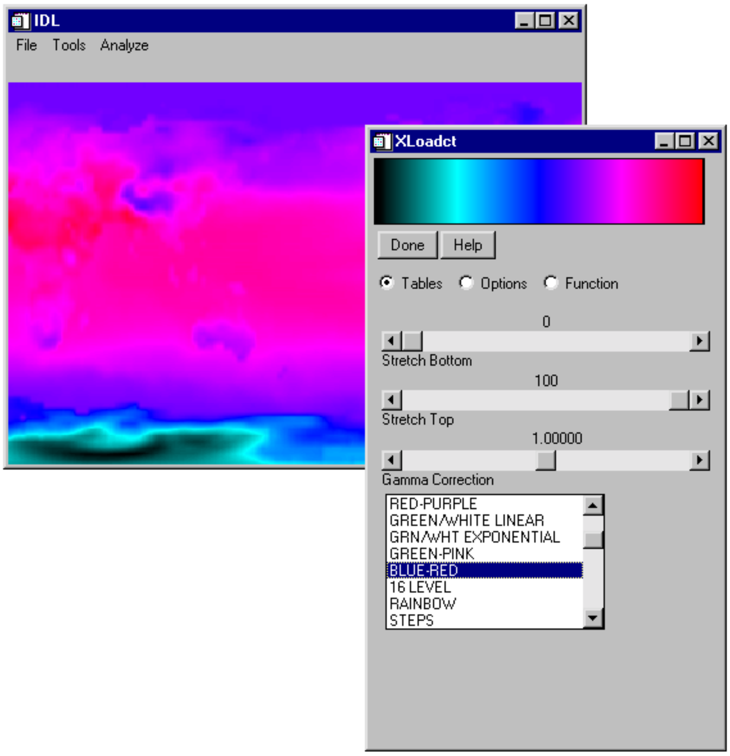


Figure 14-4: The example GUI application

Widget Types

Here is a quick description of the widget types you can create using the IDL GUIBuilder toolbar:

Widget	Description
Base	Creates a container for a group of widgets within a top-level base container (which is contained in the IDL GUIBuilder window).
Button	Creates a push button. The easiest way to allow a user to interact with your application is through a button click.
Radio Button	Creates a toggle button that is always grouped within a base container.
Checkbox	Creates a checkbox, which you can use either as a single toggle button to indicate a particular state is on or off or as a list of choices from which the user can select none to all choices.
Text	Creates a text widget.
Label	Creates a label.
Horizontal and Vertical Sliders	Creates a slider with a horizontal or vertical layout.
Droplist	Creates a droplist widget, which you can use to present a scrollable list of items for the user to select from.
Listbox	Creates a list widget, which you can use to present a scrollable list of items for the user to select from.
Draw Area	Creates a draw area, which you can use to display graphics in your application.
Table	Creates a table widget, which you can use to display data in a row and column format.
Tab	Creates a tab widget on which different “pages” (base widgets and their children) can be displayed by selecting the appropriate tab.

Table 14-1: Widget Types

Widget	Description
Tree	Creates a tree widget, which presents a hierarchical view that can be used to organize a wide variety of data structures and information.

Table 14-1: Widget Types

Note

The Select Cursor button returns the cursor to its standard state, and it indicates that the cursor is in that state. After you click on another button and create the selected widget, the cursor returns to the selection state.

Widget Properties

For each widget type, there is a set of attribute values and a set of event values you can set using the IDL GUIBuilder Properties dialog. When you select a widget in the IDL GUIBuilder window or in the Widget Browser, the Properties dialog is updated to contain the properties for the selected widget. These properties include those common to all widgets and those specific to the selected widget.

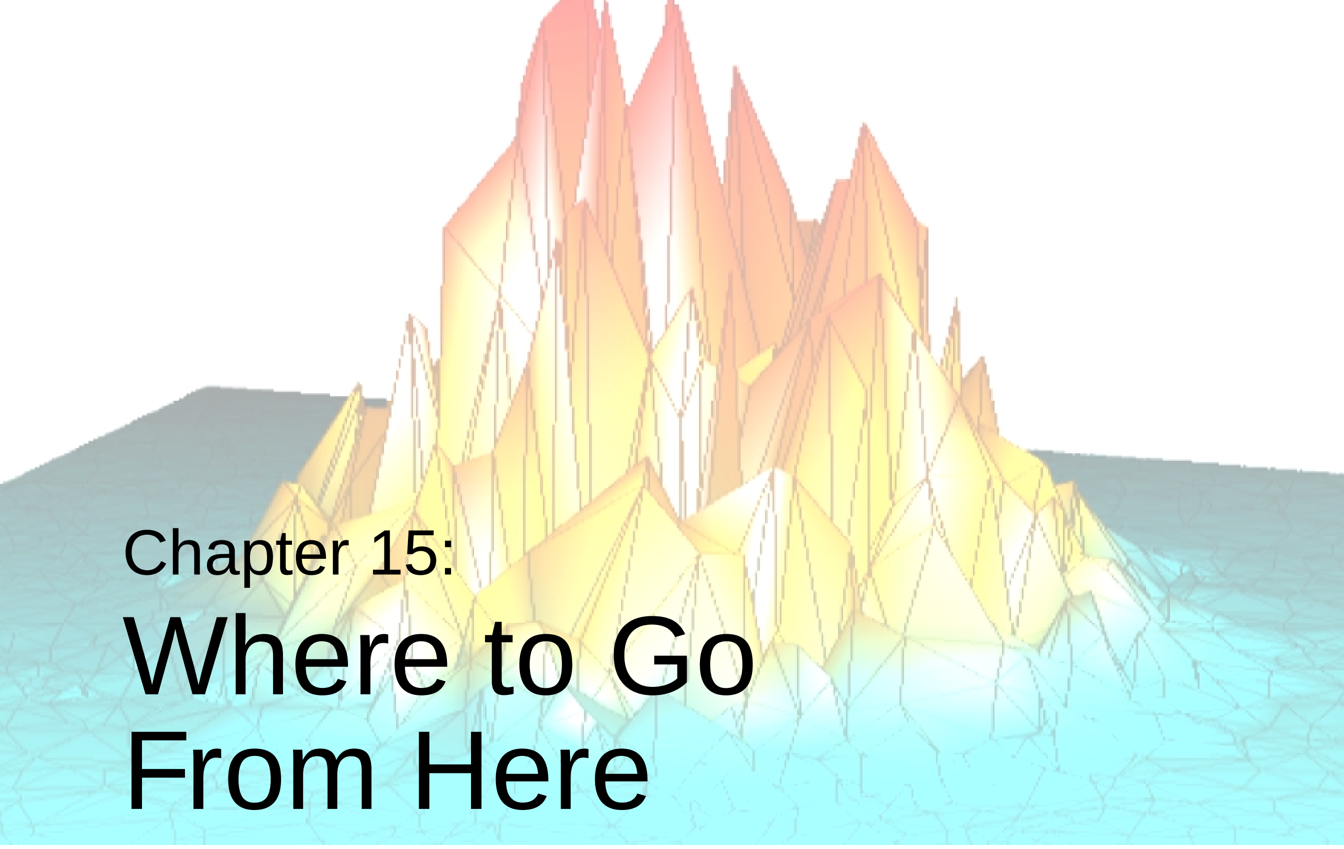
On the Attributes tab of the Properties dialog (right click on the draw area, and choose Properties. This action opens the Properties dialog for the draw area) the attributes are set to default values and are arranged in the following order:

- The Name property.
- An alphabetical list of common and widget-specific properties, combined.

On the Events tab, the possible events for a widget are listed in alphabetical order, with the common and the widget-specific events combined. By default, no event values are set initially. When you enter a routine name for an event property, you are responsible for making sure that event procedure exists. IDL does not validate the existence of the specified routine.

More Information on the IDL GUIBuilder

For more information on the IDL GUIBuilder, see the *Building IDL Applications* manual.



Chapter 15:

Where to Go From Here

Using just a few examples, you've now gained a brief glimpse at the many and powerful options that IDL offers you for data analysis, visualization, and cross-platform application development. But the power of IDL has only begun and IDL offers you many options to aid you in learning more and more about its incredible functionality.

Learning More about IDL

A multitude of resources are made available to you to assist you as you learn the many capabilities of IDL. IDL manuals are offered both printed and online (in PDF version). IDL also includes an online help system directly accessible from the IDL Development Environment.

IDL Documentation Set

If you have just purchased IDL, you will receive part of the documentation set (depending upon which products you have purchased) in printed form with your order. All IDL manuals are available in PDF form on your product CD-ROM. For more information, see "Online Manuals" on page 194.

Installing and Licensing IDL 5.6

Installing and Licensing IDL 5.6 describes how to install and license IDL on your platform. It provides information about the different types of licensing available for IDL and how to manage licensing on your system.

Using IDL

Using IDL explains IDL from an interactive user's point of view. It contains information about the IDL environment, the structure of IDL, and how to use IDL to analyze and visualize your data.

Building IDL Applications

Building IDL Applications explains how to use the IDL language to write programs - from simple procedures to large, complex applications. It contains information on the structure of the IDL language, programming techniques, and tools you can use to create applications in IDL.

Image Processing in IDL

Image Processing in IDL introduces you to the full image processing power of IDL, describing how to display, manipulate, and extract information from images. This manual features both Direct Graphics and Object Graphics examples that will aid in developing IDL applications that require image processing.

IDL Quick Reference

The *IDL Quick Reference* provides quick access to the following: IDL procedures and functions (categorized functionally and alphabetically), objects, executive commands, and statements.

IDL Reference Guide

The *IDL Reference Guide* contains detailed information about all of IDL's procedures, functions, objects, system variables, and other useful reference materials. It also contains detailed information about IDL's routines for dealing with Common Data Format (CDF), Hierarchical Data Format (HDF), Earth Observing System extensions to HDF (HDF-EOS), and Network Common Data Format (NetCDF) files.

External Development Guide

The *External Development Guide* explains how to use IDL to develop applications that interact with programs written in other programming languages.

Obsolete IDL Features

Obsolete IDL Features describes routines that have become obsolete by enhancements to the IDL language. While these routines continue to exist, RSI recommends that you do not use routines that have become obsolete in new code.

IDL Master Index

The *Master Index* is a combined and comprehensive index covering all manuals in the IDL documentation set.

Note

Additional documentation can be ordered from Research Systems by contacting RSI Sales at (303) 786-9900 or by visiting `www.ResearchSystems.com`.

IDL DataMiner Guide

The *IDL DataMiner Guide* contains information on using IDL to interact with databases using the Open Database Connectivity (ODBC) interface.

Online Manuals

All volumes of the IDL documentation set are also available in Adobe Acrobat Portable Document Format (PDF). These PDF files are automatically installed on your machine with IDL. Remember that in order to view these manuals, you will need a copy of Adobe's Acrobat Reader with Search software (version 3.0 or later). A copy of Adobe Acrobat Reader is included on your product CD-ROM. For more information on Adobe Acrobat Reader, visit their World Wide Web site at `www.adobe.com`.

How to Access IDL Online Manuals

To access the IDL online manuals after you have installed IDL:

On Windows, select **Start** → **Programs** → **Research Systems IDL 5.6** → **IDL Online Manuals & Tutorials**.

On UNIX and Mac OS X, execute the following at the prompt:

```
idlman
```

The IDL online manuals can also be found in the `info` directory of your product CD-ROM.

Navigation of the IDL Online Manuals

The online IDL manuals are fully hypertext linked for easy navigation. An Online Guide (onlguide.pdf) file is also included which is your guide to the IDL documentation set. It has links for all manuals in the documentation set as well as links on how to get more information from Research Systems.

Searching within the Online Manual Set

The IDL online manuals are set up to search for any information you might need within the IDL manual set. To search the IDL manual set, you can click on the binocular/page button in the Acrobat Reader tool bar after you have opened any IDL manual in the set including the Online Guide.

Online Help

IDL is equipped with extensive on-line help facilities that provide two kinds of information: documentation of IDL procedures, functions, and keywords, and information on the status of the IDL environment. There are several ways to access these help facilities from within the IDL Development Environment.

The IDL Development Environment Help Menu

One way is to start Help by selecting “Contents” from the “Help” pull-down menu in the IDL Development Environment. You can also search using the “Find Topic” selection on the same menu.

The Question Mark

You can access the IDL Help by entering a question mark (?) at the IDL prompt. The IDL Online Help window appears. The most current documentation on any aspect of IDL is available through this command. Although the help window has buttons for performing searches, you can also perform a keyword search from the command line by entering “?” followed by a keyword for which you want to search. For example, to search for topics related to contouring when starting the help system, you could enter:

```
? CONTOUR
```

IDL Help Outside of the IDL Development Environment

You may also access the IDL Help system outside of the IDL Development Environment.

On UNIX and Mac OS X, execute the following at the prompt:

```
idlhelp
```

HELP Command

The HELP procedure gives information about the IDL session. Enter:

```
HELP
```

with no additional parameters to display an overview of the current IDL session including one-line descriptions of all variables and the names of all compiled procedures and functions. Enter:

```
HELP, variable
```

to display information about that variable’s type. Many keyword parameters can be used with the HELP procedure to retrieve more specific information.

IDL Demo Applications and Examples

The IDL Demo Applications illustrate some of the many ways IDL can help visualize data. The IDL Demo Applications are a series of programs written in the IDL language that demonstrate different aspects of IDL. To Start the Demo Applications, complete the following steps:

For Windows, by clicking the Windows Start button and selecting **Start** → **Programs** → **Research Systems IDL 5.6** → **IDL Demo**.

For UNIX and Mac OS X, enter the following at the UNIX prompt:

```
idldemo
```

Note

If you have already started IDL, you can simply type in DEMO at the IDL prompt.

```
IDL> DEMO
```

Another way to access the IDL Demo System is to use the “Run Demo” toolbar button on the IDLDE toolbar. To use this feature simply click the button and the dialog for running the demo will appear.

IDL also comes with many built-in examples such as source code and example data files. These can be found in the *`RSI-DIR/`*`examples` where *`RSI-DIR`* is the directory in which you have installed IDL.

Contacting RSI

Address

Research Systems, Inc.
4990 Pearl East Circle
Boulder, CO 80301

Phone

(303) 786-9900
(303) 413-3920 (Technical support)

Fax

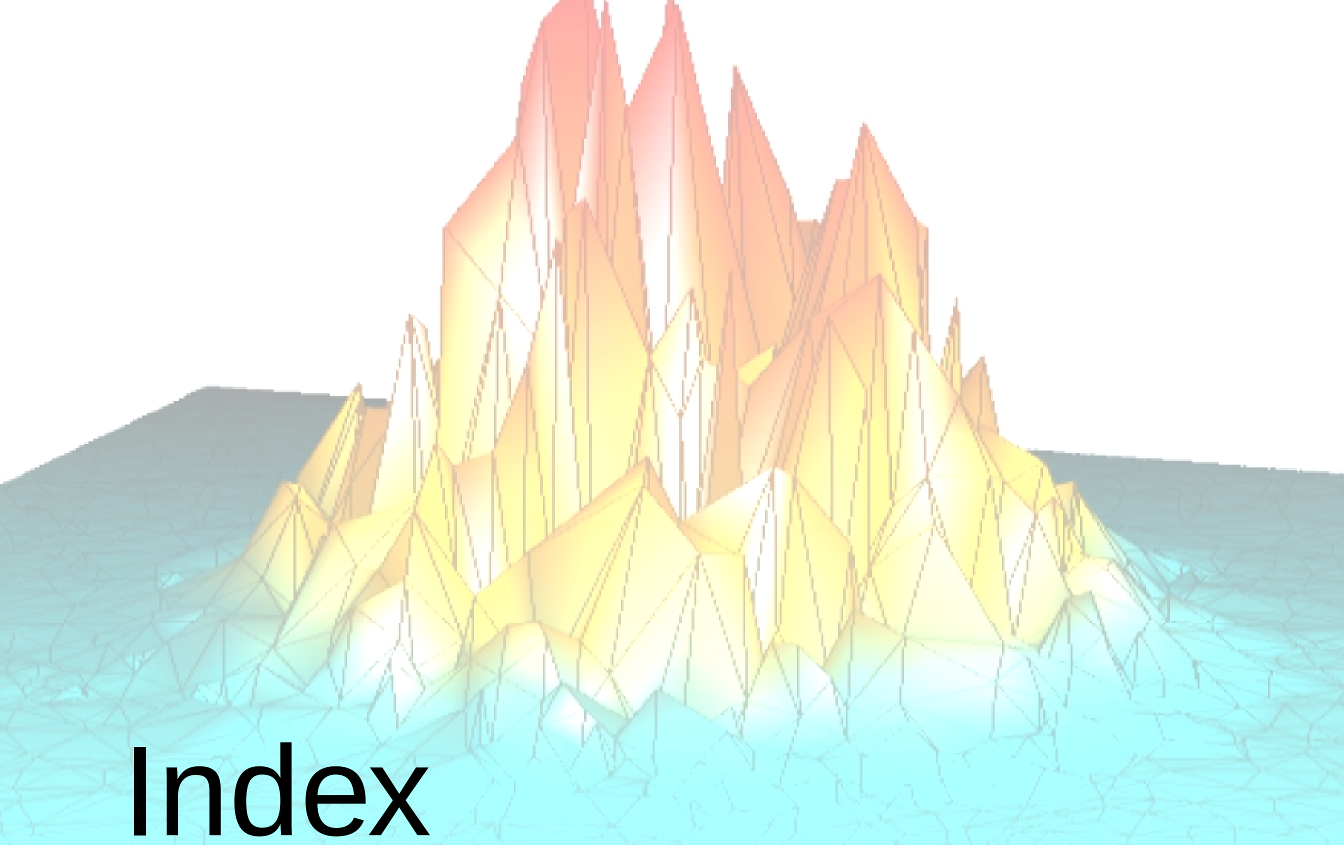
(303) 786-9909

E-mail

Sales inquiries: `info@ResearchSystems.com`
Technical support: `support@ResearchSystems.com`
Training information: `training@ResearchSystems.com`

World Wide Web

Visit Research Systems’ web site at: Visit Research Systems’ web site at `www.RSInc.com`.



Index

Symbols

$, continuation character, 144
&, multiple command character, 144

Numerics

2D plot, 51

A

abbreviating keywords, 115
Adobe Acrobat Portable Document Format (PDF), 194
animation
  wire mesh surface, 143
annotating
  maps, 117
axes
  turning off, 146
azimuthal equidistant map projection, 116

B

base widgets
  using, 187
button widgets
  using, 187
byte scaling, 64, 89

C

checkboxes
  using, 187

code
  IDL GUIBuilder generated, 181
  modifying generated, 182
color tables
  example, 184
  loading, 88
colors
  byte scaling, 89
  contrast enhancement, 68
  filling contours, 93
  scaling, 64
Command Input Line
  IDLDE for Motif, 18
Contents button, 195
contours, 91
  filling, 93
  levels, 92
  3D perspective, 94
  tickmarks, 93
contrast enhancement, 68
creating data, 51, 85
customizing routines, 96

D

data
  irregularly-gridded, 130
data sets
  creating, 51
  plotting, 44
datasets
  creating, 51, 85, 131
differentiated smoothing, 76
dismissing windows, 64
draw widgets
  example application using, 179
  using, 187
drawing latitude and longitude lines, 115
droplist widgets
  using, 187

E

efficiency
  vector and array operations, 169
elevation levels. contours, 92
events
  handling in IDL GUIBuilder code, 182

F

files
  IDL GUIBuilder
    generated, 181
  modifying generated, 182
  reading image data, 63
filling contours, 93
filtering
  frequency domain, 55
filtering techniques, 55
Fourier transform filtering, 55
frequency domain filtering, 55

G

globe, drawing, 115
graphics
  window resizing, 67
  windows, dismissing, 64
gridding
  latitude and longitude lines, 115
  quintic, 135

H

help on IDL, 195

I

IDL
  Code Profiler, 159

getting help, 195
IDL Development Environment
Layout, 17
IDL GUIBuilder
base widgets, using, 187
button widgets, using, 187
checkbox widgets, using, 187
color table example, 184
compiling and running example, 185
creating draw area, example, 179
defining menus, example, 177
draw widgets, using, 187
droplists, using, 187
generating code, 181
horizontal slider, using, 187
label widgets, using, 187
listbox widgets, using, 187
modifying code, example, 182
radio button widgets, using, 187
smooth example, 184
starting, 177
table widgets, using, 187
test mode, 181
text widgets, using, 187
toolbar, 176
tools, 176
vertical slider, using, 187
writing event-handling code, 182
images
contrast enhancement, 68
displaying, 64, 64
opening, 63
reading, 63, 63
rotating, 79
sharpening, 74
smoothing, 74
input and output, 24
interpolation
linear, 135
quintic, 135
irregularly-gridded data, 130
iso-surfaces, 10

K

keywords
abbreviating, 115

L

label widgets
using, 187
latitude lines, 115
levels, contour, 92
lines
drawing latitude and longitude, 115
horizon, 115
listbox widgets
using, 187
LIVE_PLOT
using, 46
LIVE_PLOT procedure, 46
longitude lines, 115

M

mapping, 112
annotations, 117
grid of latitude and longitude lines, 115
horizon line, 115
limiting region mapped, 116
projection
azimuthal equidistant, 116
orthographic, 115
multiple plot displays, 58

O

opening
image files, 63

orthographic map projection, 115
Output Log
  IDLDE for Motif, 19
overplotting, 93

P

PDF files, 194
performance
  analyzing, 159
pixel
  brightness profile, 80
pixels
  scaling, 71
plotting
  annotating maps, 117
  contours, 91
  displaying multiple plots, 58
  irregularly-gridded data, 130
  other capabilities, 45
  overplotting, 93
  simple, 41
  surfaces, 84, 86, 152
  symbols, 132
  text size, 88
  titles, 54, 88
plotting with data sets, 44
printing a plot, 43
profiling, 159
projections, map
  azimuthal equidistant, 116
  orthographic, 115
projects
  overview, 18, 160

Q

quintic interpolation, 135
quitting IDL, 20

R

radio button widgets
  using, 187
reading
  images, 63
reading binary data, 31
reading images, 36
real-world test data, 52
resizing
  graphics windows, 67
rotating
  images, 79
routines
  customizing, 96

S

saving a template, 35
scaling pixels, 71
scientific data formats, 25
sharpening images, 74
signal processing
  and IDL, 50
SIN function, 51
sinewave function, 51
slices
  making, 106
slider widgets
  using, 187
smoothing
  example, 184
smoothing images, 74
standard
  image file formats, 25
Status Bar
  IDLDE for Motif, 19
surfaces
  constant-density, 10
  displaying, 108
  iso-surfaces, 10

plotting, 84, 86, 152
wire mesh animation, 143

T

table widgets
using, 187
test data, 52
test mode, IDL GUIBuilder, 181
text
size, 88
text widgets
using, 187
three-dimensional
contour plot, 94
surfaces, 86
thresholding, 68
toolbars
IDL GUIBuilder, 176
IDLDE for Windows, 17
tutorials, 12

U

unsharp masking, 75

V

Variable Watch Window, 158
variables
displaying current, 158

W

widgets
bases, using, 187
buttons, using, 187
checkboxes, using, 187
creating in IDL GUIBuilder, 176
draw, using, 187
droplists, using, 187
labels, using, 187
listbox, using, 187
radio buttons, using, 187
slider, using, 187
table, using, 187
text, using, 187
windows
deleting, 148
dismissing, 64
resizing, 67
wire mesh surface
animation, 143
writing binary data, 31
writing images, 36