Introduction to IDL

László Nyúl

Department of Image Processing and Computer Graphics
University of Szeged
Hungary

Ever Programmed Before?

- Yes: it may become useful
- No: it may become useful

The Power of IDL

- IDL is a complete computing environment for the interactive analysis and visualization of data.
- IDL integrates a powerful, array-oriented language with numerous mathematical analysis and graphical display techniques.
- You can explore data interactively using IDL commands and then create complete applications by writing IDL programs.

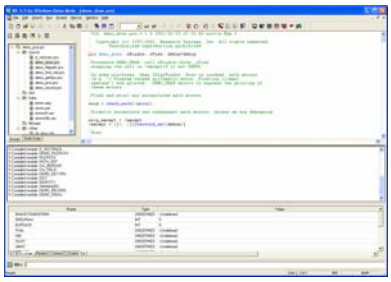
Advantages of IDL

- Operators and functions work on entire arrays (without using loops), simplifying interactive analysis and reducing programming time.
- Immediate compilation and execution of IDL commands provides instant feedback and hands-on interaction.
- Rapid 2D plotting, multi-dimensional plotting, volume visualization, image display, and animation allow you to observe the results of your computations immediately.
- Many numerical and statistical analysis routines—including Numerical Recipes routines—are provided for analysis and simulation of data.

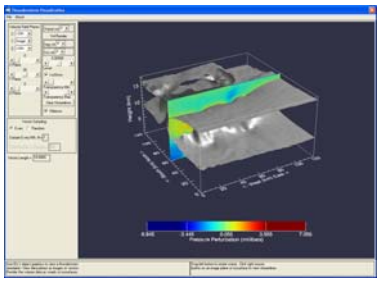
Advantages of IDL

- IDL's flexible input/output facilities allow you to read any type of custom data format. Support is provided for common image standards (BMP, Interfile, JPEG, PNG, PPM, TIFF), scientific data formats (CDF, HDF), and other data formats (ASCII, Binary, DICOM, DXF, WAV).
- IDL widgets can be used to quickly create multi-platform graphical user interfaces to your IDL programs.
- IDL programs run the same across all supported platforms (UNIX, VMS, Microsoft Windows, and Macintosh systems) with little or no modification.
- Existing FORTRAN and C routines can be dynamically-linked into IDL to add specialized functionality. Alternatively, C and FORTRAN programs can call IDL routines as a subroutine library or display engine.

IDLDE



IDL Demo Application



IDL-Specific Programming Language Constructs

- ;
- $
- &
- Vector[4:12]
- Vector[4:*]
- Array[*, j]
- Array[array2]
- A < B, C > D
- EQ, NE, LT, LE, GT, GE
- X # Y, A ## B
- !PI
- AND, OR, NOT, XOR

Arrays

```
array = [1.0, 2.0, 3.0, 4.0, 5.0]
array = [[1, 2, 3], [4, 5, 6]]
PRINT, array
array = FINDGEN(3, 4, 5)
array = array/2

array = [1, 2, 3]
new = array[1]
PRINT, new ; prints 2
```

Structures

```
A = {STAR, NAME: '', RA: 0.0, DEC: 0.0, INTEN: FLTARR(12)}
A = {NAME: '', RA: 0.0, DEC: 0.0, INTEN: FLTARR(12)}
A = CREATE_STRUCT('NAME', '', 'RA', 0.0, 'DEC', 0.0, 'INTEN',
   FLTARR(12))

A = {STAR, NAME: 'SIRIUS', RA: 30., DEC: 40., $
   INTEN: INDGEN(12)}
A.NAME = 'BETELGEUSE'
PRINT, A.NAME, A.RA, A.DEC
Q = A.INTEN[5]
A.RA = 23.31
A.INTEN = 0
B = A.INTEN[3:6]
A.NAME = 12
B = A
```

A Short IDL Command Session

```
IDL> PRINT, 3*5
      15
IDL> A=3*5
IDL> HELP, A
A               INT       =       15
IDL> A=SQRT(A) & HELP, A
A               FLOAT     =       3.87298
IDL> A = [1, 2, 3, 4, 5, 6]
IDL> PRINT, A, 2*A
       1       2       3       4       5       6
       2       4       6       8      10      12
IDL> B = SQRT(A)
IDL> HELP, A, B
A               INT       = Array[6]
B               FLOAT     = Array[6]
IDL> PRINT, B
      1.00000      1.41421      1.73205      2.00000
   2.23607      2.44949
```

Reading and Writing Images

- Compound Widgets and Dialogs
  - CW_FILESEL
  - **DIALOG_PICKFILE**
  - DIALOG_READ_IMAGE
  - DIALOG_WRITE_IMAGE
- Images
  - QUERY_* (BMP, JPEG, PICT, PNG, PPM, SRF, TIFF)
  - QUERY_DICOM
  - **READ_*** (BMP, JPEG, PICT, PNG, PPM, SRF, TIFF, X11_BITMAP, XWD)
  - **READ_DICOM**
  - READ_INTERFILE
  - WRITE_* (BMP, JPEG, NRIF, PICT, PNG, PPM, SRF, TIFF)

Imaging Routines

- TV
  - displays images on the image display
- TVSCL
  - scales the intensity values of the image into the range of the display device, then displays the result on the image display
- TVLCT
  - loads a new color table into the display device
- TVRD
  - reads image pixels back from the display device

Mathematics

- Arrays and Matrices
- Correlation Analysis
- Curve and Surface Fitting
- Eigenvalues and Eigenvectors
- Gridding and Interpolation
- Hypothesis Testing
- Derivation and Integration
- Linear Systems
- Nonlinear Equations
- Optimization
- Sparse Arrays
- Time Series Analysis
- Multivariate Analysis

Signal Processing Routines

- Interpolation
  - **INTERPOL**
- Convolution
  - **CONVOL**
  - BLK_CON
- Transforms
  - **FFT**
  - WTN
- Filters
  - **DIGITAL_FILTER**
  - SAVGOL
  - **MEDIAN**
  - SMOOTH
  - HANNING
  - HILBERT

Array Creation Routines

- **INDGEN**
  - Return an integer array with each element set to its subscript (prefix with B, C, DC, D, F, L, S for other element types, e.g. FINDGEN)
- **ARR**
  - Create a vector or array (prefix with BYT, COMPLEX, DBL, DCOMPLEX, FLT, INT, LON, OBJ, PTR, STR for different element types, e.g., FLTARR)
- IDENTITY
  - Return an identity array
- MAKE_ARRAY
  - General purpose array creation
- REPLICATE
  - Form array of given dimensions filled with a value

Array Manipulation Routines

- 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

Array and Image Processing Routines

- **CONGRID**
  - Resample image to any dimensions
- **MAX**
  - Return the maximum element of an array
- **MEDIAN**
  - Median function and filter
- MIN
  - Return the minimum element of an array
- REBIN
  - Resample array by integer multiples
- SIZE
  - Return array size and type information
- **TOTAL**
  - Sum array elements
- **WHERE**
  - Return subscripts of non-zero array elements

Avoiding IF Statements by Summing Elements

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

A = A + (B GT 0) * B

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

Avoiding IF Statements by Using Array Operators and WHERE

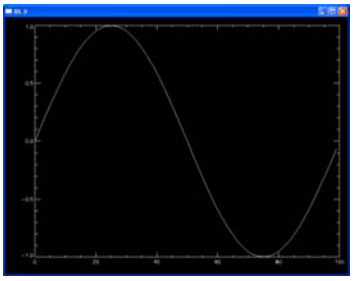
```
FOR I=0,(N-1) DO IF A[I] LE 0 THEN
  C[I]=-SQRT(-A[I]) ELSE C[I]=SQRT(A[I])

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

NEGS = WHERE(A LT 0)
C = SQRT(ABS(A))
C[NEGS] = -C[NEGS]
```

Using Vector and Array Operations

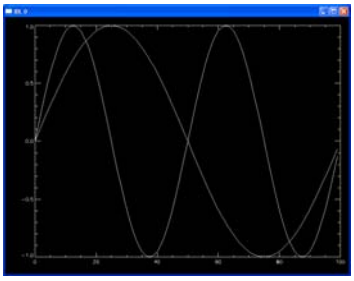
```
FOR I = 0, 511 DO FOR J = 0, 255 DO BEGIN
TEMP=IMAGE[I, J]
image[I, J] = image[I, 511 - J]
image[I, 511-J] = temp
ENDFOR

FOR J = 0, 255 DO BEGIN
temp = image[*, J]
image[*, J] = image[*, 511-J]
image[*, 511-J] = temp
ENDFOR

image2 = BYTARR(512, 512)
FOR J = 0, 511 DO image2[*, J] = image[*, 511-J]

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

image = ROTATE(image, 7)

image = REVERSE(image, 2)
```

Simple Plotting

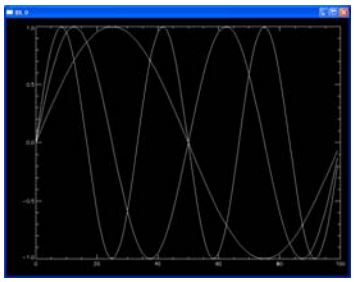
X= 2*!PI/100 * FINDGEN(100)
PLOT, SIN(X)



OPLOT, SIN(2*X)

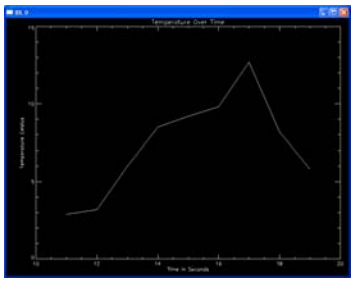


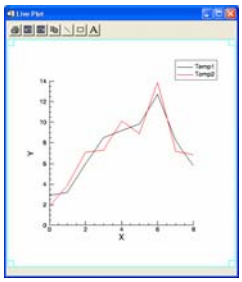
OPLOT, SIN(3*X)



Plotting with Data Sets

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

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

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

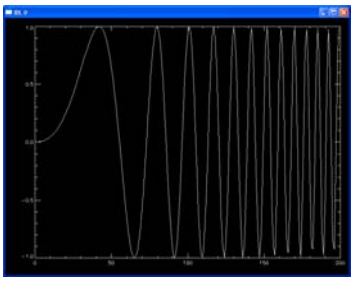




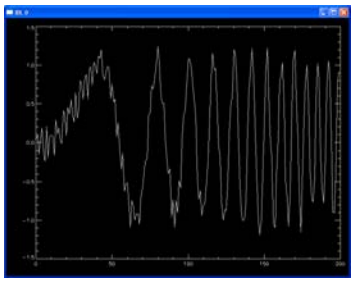
Signal Processing

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

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

SMOOTHED=SMOOTH(NOISY, 5)
```

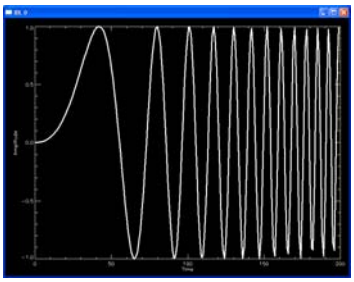
PLOT, ORIGINAL



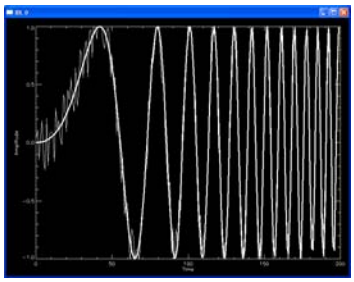
PLOT, NOISY



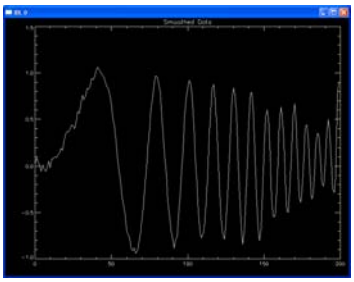
PLOT, ORIGINAL, XTITLE="Time", YTITLE="Amplitude", THICK=3



OPLOT, NOISY



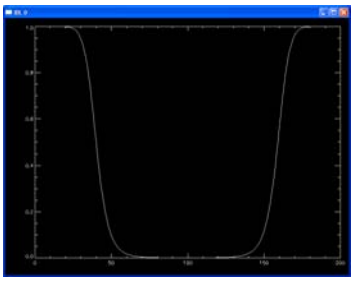
PLOT, SMOOTHED, TITLE='Smoothed Data'



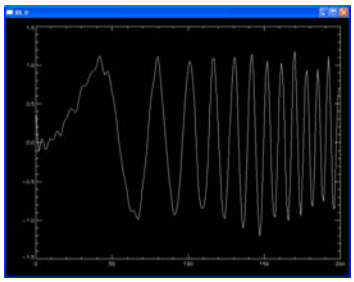
Signal Processing

```
Y=[FINDGEN(100),FINDGEN(100)-100]
Y[101:199]=REVERSE(Y[0:98])
filter=1.0/(1+(Y/40)^10)

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

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

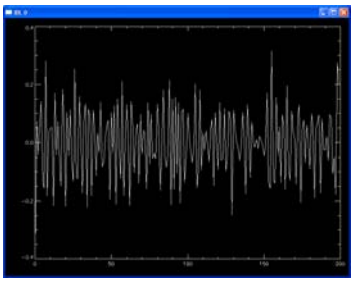
PLOT, FILTER



PLOT, LOWPASS



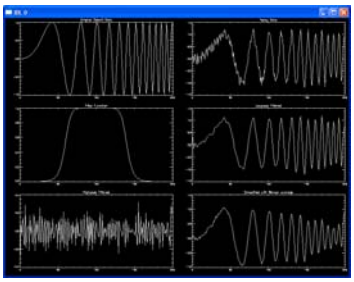
PLOT, HIGHPASS



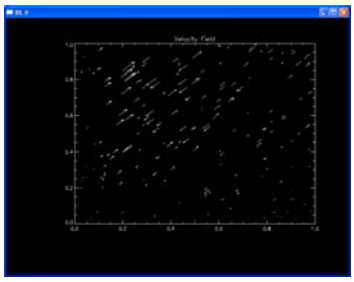
Multiple Plots in One Window

```
!P.MULTI=[0,2,3]
PLOT,ORIGINAL,TITLE='Original (Ideal) Data'
PLOT,NOISY,TITLE='Noisy Data'
PLOT,SHIFT(filter,100),TITLE='Filter Function'
PLOT,LOWPASS,TITLE='Lowpass Filtered'
PLOT,HIGHPASS,TITLE='Highpass Filtered'
PLOT,SMOOTHED,TITLE='Smoothed with Boxcar average'

!P.MULTI = 0
```

Multiple Plots in One Window



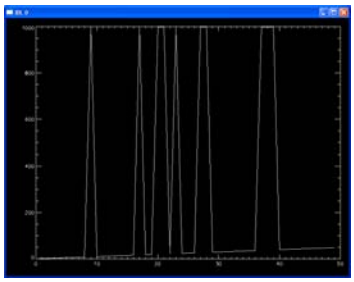
VX = ORIGINAL # FINDGEN(200)
VY = NOISY # FINDGEN(200)
VEL, VX, VY



Plotting Noisy or Missing Data

```
A = INDGEN(50)
A[RANDOMU(SEED, 10) * 50] = 999
```

PLOT, A



PLOT, A, MAX_VALUE=998

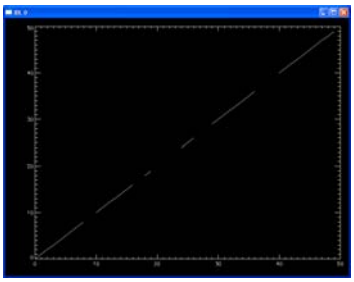
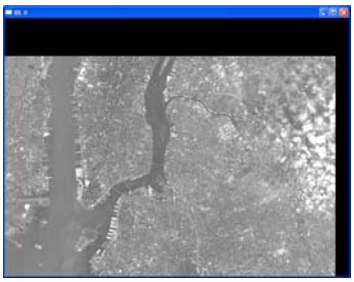


Image Processing

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

WDELETE
```

TV, MYIMAGE



TVSCL, MYIMAGE

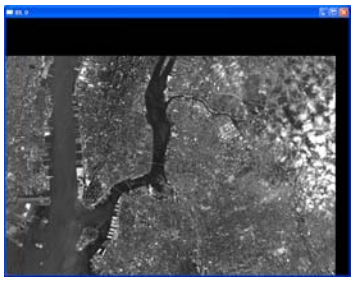
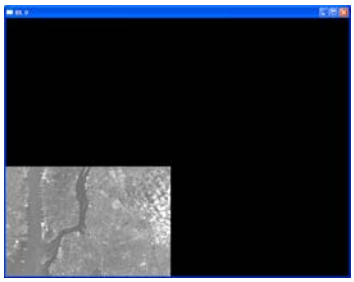
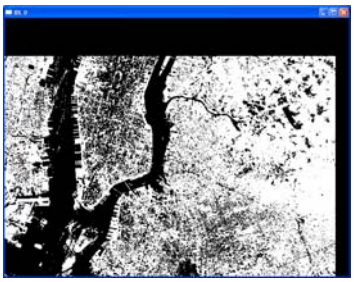


Image Processing

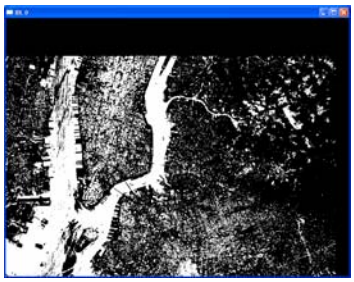
NEWIMAGE=REBIN(MYIMAGE,384,256)
TV, NEWIMAGE



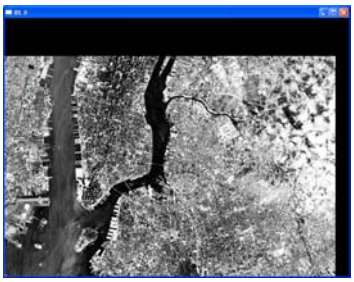
TVSCL, MYIMAGE GT 140



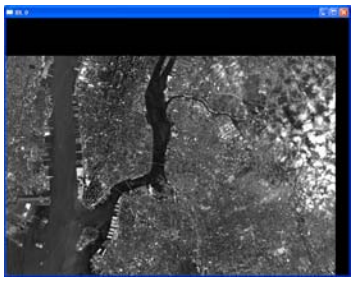
TVSCL, MYIMAGE LT 140



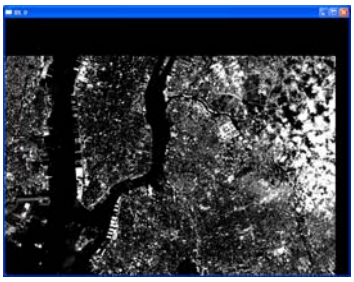
TV, HIST_EQUAL(MYIMAGE)



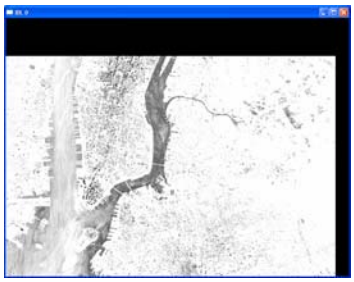
TVSCL, MYIMAGE > 100



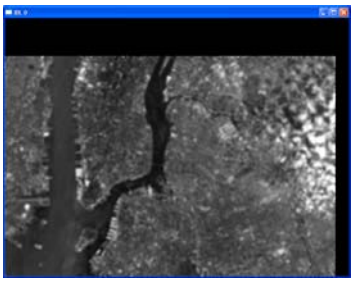
TVSCL, MYIMAGE < 140



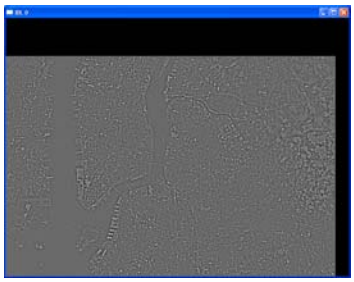
TV, BYTSCL(MYIMAGE,MIN=140, MAX=200,TOP=!D.TABLE_SIZE)



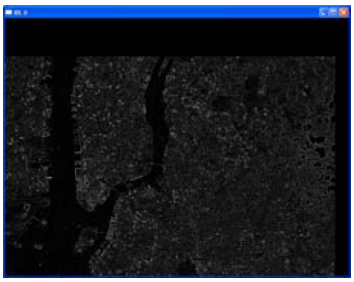
TVSCL, SMOOTH(MYIMAGE,7)



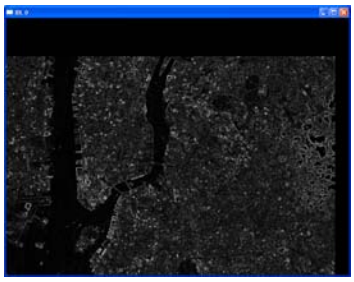
TVSCL, FLOAT(MYIMAGE)-SMOOTH(MYIMAGE,7)



TVSCL, ROBERTS(MYIMAGE)



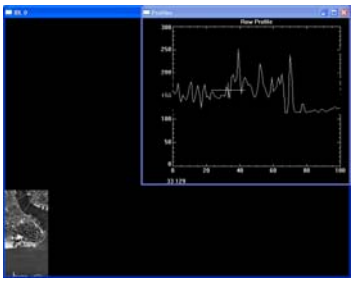
TVSCL, SOBEL(MYIMAGE)



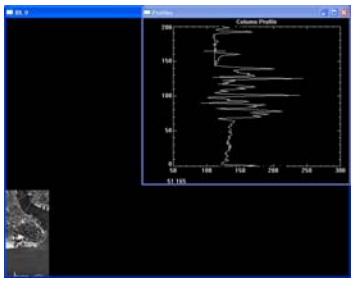
E = MYIMAGE[100:300, 150:250]
ERASE & TV, E

TVSCL, ROTATE(E,1)

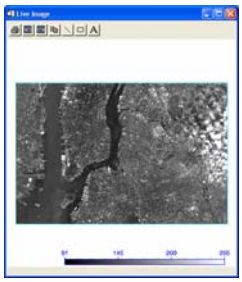
PROFILES, R



PROFILES, R



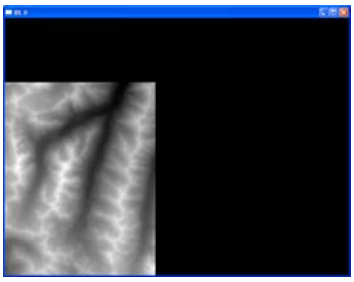
LIVE_IMAGE, MYIMAGE



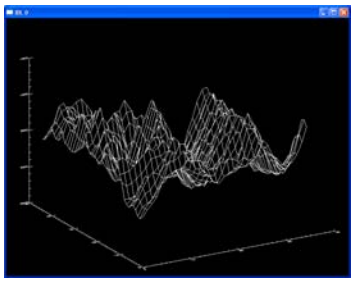
Surface and Contour Plotting

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

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

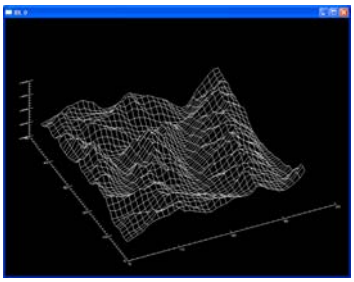
TVSCL, MARBELLS_BINARY.A



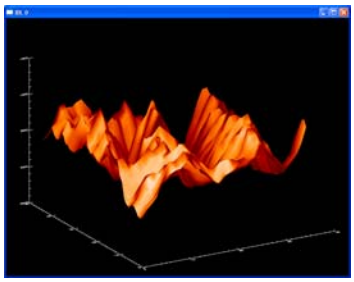
SURFACE, A



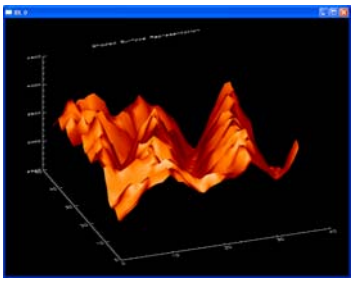
SURFACE, A, AX = 70, AZ = 25



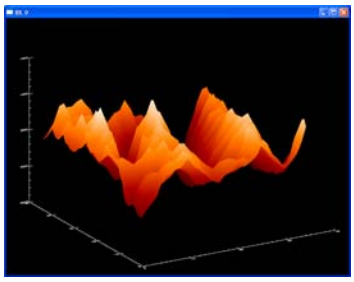
LOADCT, 3
SHADE_SURF, A



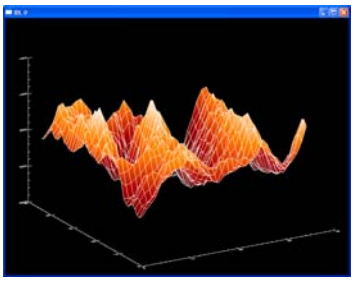
SHADE_SURF,A,AX=45,AZ=20,
CHARSIZE=1.5,TITLE='Shaded …'



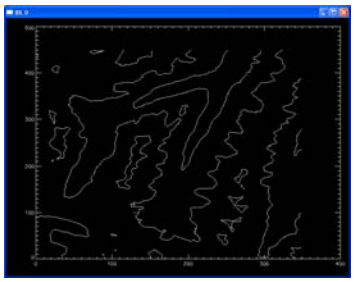
SHADE_SURF, A,
SHADE=BYTSCL(A)



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

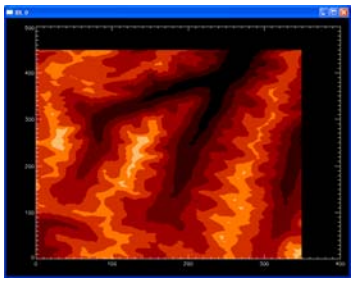


A=MARBELLS_BINARY.A
CONTOUR, A

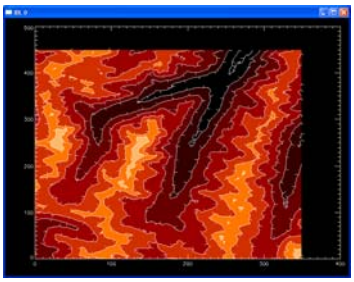


CONTOUR, A, NLEVELS=8,
C_LABELS=[0,1]

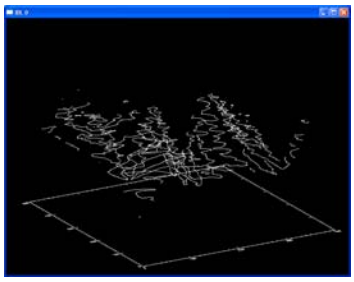
CONTOUR,A,NLEVELS=8,/FILL



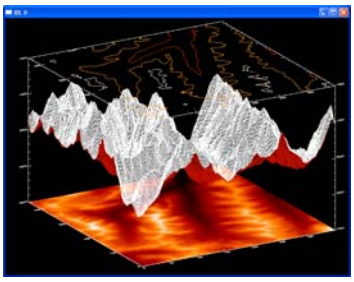
CONTOUR, A, NLEVELS=8,
/OVERPLOT, /DOWNHILL



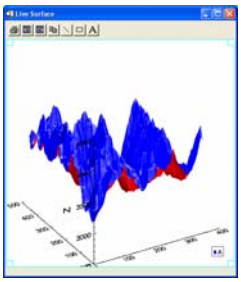
SURFR
CONTOUR,A,NLEVELS=8,/T3D



SHOW3, A



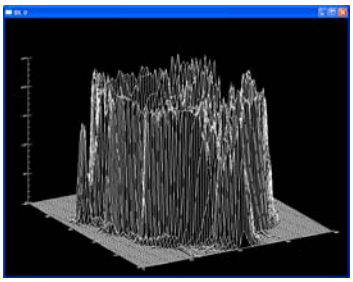
LIVE_SURFACE, A



Volume Visualization

```
MYTEMPLATE=BINARY_TEMPLATE(FILEPATH('head.dat',
   SUBDIR=['examples', 'data']))
HEAD_BINARY=READ_BINARY(FILEPATH('head.dat',
   SUBDIR=['examples','data']),TEMPLATE=MYTEMPLATE)
SLICE=(HEAD_BINARY.B)[*,*,25]
```

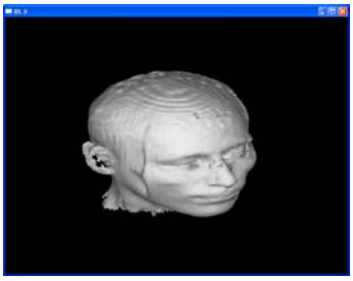
SURFACE, SLICE



Volume Visualization

```
SHADE_VOLUME,HEAD_BINARY.B,70,V,P,/LOW
SCALE3,XRANGE=[0,80],YRANGE=[0,100],ZRANGE=[0,57]
```

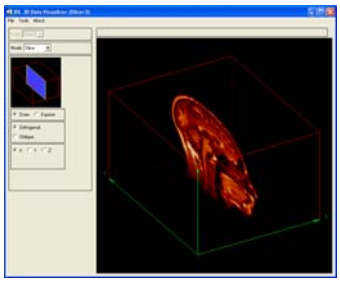
TV,POLYSHADE(V,P,/T3D)



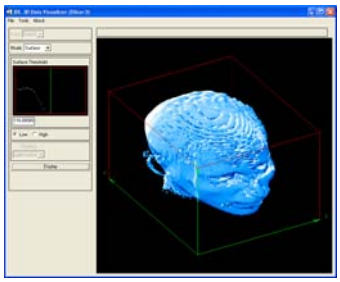
Volume Visualization

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

SLICER3,BDATA “Slice Mode”



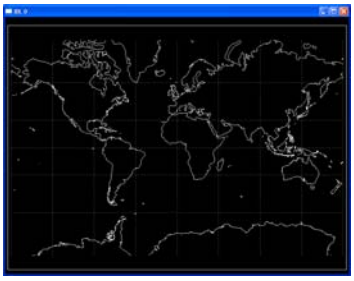
SLICER3,BDATA “Surface Mode”



remove_bridges.pro

```
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
```

MAP_SET, /GRID, /CONTINENTS, /MERCATOR



Using the IDL GUIBuilder

- Widgets are simple graphical objects
- Attribute properties control the display, initial state and behavior of widgets
- Event properties (set of events to which it can respond)
- Event procedure name

- Interface design (*.prc)
- Widget definition code (*.pro)
- Event handling code (*_eventcb.pro)

Widget Types

- Base
- Button
- Radio Button
- Checkbox
- Text
- Label
- Horizontal and Vertical Sliders
- Droplist
- Listbox
- Draw Area
- Table

Event-Handler Example

```
PRO example_event, event
CASE event.value OF
   'Quit Example' : WIDGET_CONTROL, event.top, /DESTROY
   'View an Image' : BEGIN
      path = FILEPATH('', SUB=['examples', 'data'])
      filename = DIALOG_PICKFILE(PATH=path)
      IF (STRLEN(filename) EQ 0) THEN RETURN
      OPENR, unit, filename, /GET_LUN
      fileinfo = FSTAT(unit)
      dim = SQRT(fileinfo.size)
      image = BYTARR(dim, dim)
      READU, unit, image
      FREE_LUN, unit
      SLIDE_IMAGE, REBIN(image, dim*2, dim*2), GROUP = event.top, /REGISTER, RETAIN=2
   END
ENDCASE
END

PRO wexample
base = WIDGET_BASE(/COLUMN, XPAD=10, YPAD=10)
menu = CW_GROUP(base, ['View an Image', 'Quit Example'], /COLUMN, /RETURN_NAME)
WIDGET_CONTROL, base, /REALIZE
XMANAGER, 'example', base
END
```

Final Words

- First, you may find the philosophy and the language constructs of IDL strange ☹
- After a few hours/days work, you may find yourself being an IDL guru ☺

…and there is always HELP, ?, F1, or someone to turn to…

Sources of Information

- Research Systems, Inc.
  - http://www.rsinc.com/
- David W. Fanning
  - http://www.dfanning.com/