

# ENVI/IDL

## 二次开发教程

北京星图环宇科技有限公司

王志成

zawang@imagetekinfo.com

010-62054260-142



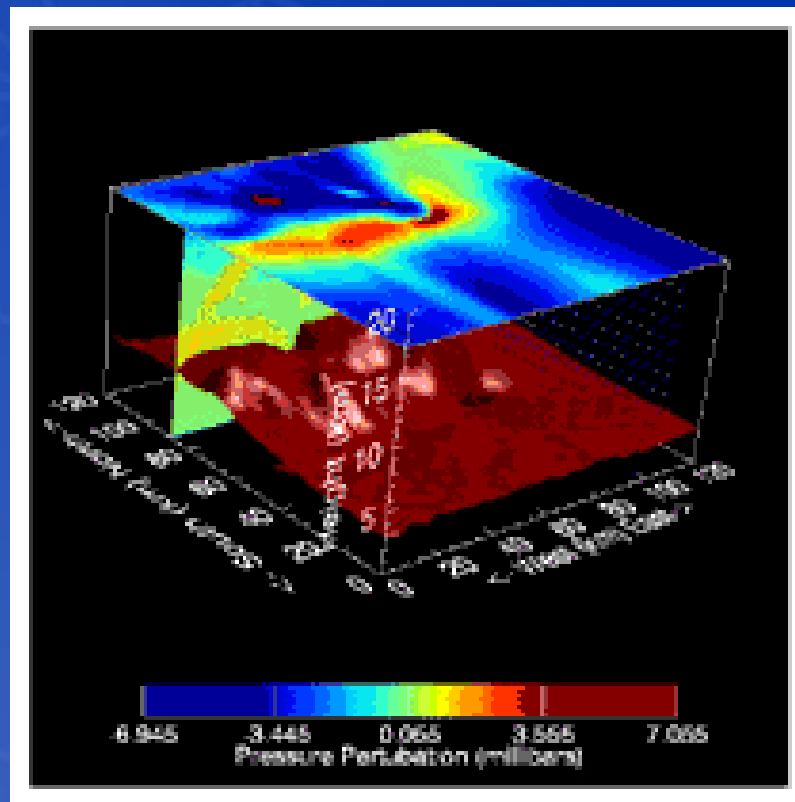
星图环宇  
ImageInfo

# 主要内容

- IDL基础
- 编写IDL程序
- ENVI/IDL二次开发介绍
- 波段和波谱运算函数
- ENVI批处理模式
- 用户函数
- ENVI提供的各种编程工具
- 综合实例

# 一、IDL基础

- IDL是进行数据分析、可视化及跨平台应用开发的最佳选择。IDL集可视、交互分析、大型商业开发为一体，为您提供最完善、最灵活最有效的开发环境。
- IDL是美国RSI公司推出的面向矩阵的第四代计算机语言。



# IDL语言的特性

- 高级图像处理能力
- 交互式二维和三维图形技术
- 面向对象的编程方式
- **OpenGL**图形加速
- 量化可视化表现
- 成数学与统计学算法
- 灵活的数据输入输出方式
- 跨平台图形用户界面工具包
- 连接**ODBC**兼容数据库
- 多种程序连接工具.....

# IDL的发展

- 在国外**IDL**已经被列为大学的标准课程，其功能和应用效果完全可以替代如**Matlab**等其他同类科学计算应用软件。
- 在国内**IDL**要比国外稍微滞后，还处在推广和应用的初期。许多科研单位和一些大学在与国外单位交流中，特别是一些留学归国人员，是**IDL**在国内应用的先行者和忠实用户。
- 随着**IDL**应用和市场的广泛进入和从科研院所的高端应用到更广泛地民用化的接受过程，越来越多的人将了解和接受应用**IDL**。

## IDL程序示例:

- **Demo\_tour**

本程序显示了IDL提供的丰富的demo程序。

## 1.1、IDL变量

- 变量定义

变量在使用前，无需说明类型。

- 变量的命名规则

变量名称必须以字母和下划线开头，可以包括字母、数字、下划线、美元符号

正确：

- `reade6_$file`
- `only_8_bit`
- `ComputerType`
- `variables`
- `_day_of_year`

错误：

- `name.last`
- `third%file`
- `4th_list`
- `$temp`

- 变量数据类型

**Byte** 字节型   **Int** 有符号整数   **UInt** 无符号整数

**Long** 有符号长整型   **Ulong** 无符号长整型   **Long64** 64位长整型

**Float** 浮点型   **Double** 双精度浮点型

**Complex** 复数   **Dcomplex** 双精度复数

**String** 字符串   0-32767个字符

**Struct** 结构

**Pointer** 指针

**Objref** 对象的引用



## •数据类型的转换

数据类型	创建变量例子	数据类型转换函数
Byte	Var=1B	thisVar=Byte(variable)
Int	Var=1	thisVar=Fix(variable)
Long	Var=1L	thisVar=Long(variable)
Long64	Var=1LL	thisVar=Long64(variable)
Uint	Var=1U	thisVar=Uint(variable)
Float	Var=1.0	thisVar=Float(variable)
Double	Var=1.0D	thisVar=Double(variable)
String	Var=""	thisVar=String(variable)
Pointer	Var=ptr_new()	
Object	Var=obj_new()	

## 1.2、IDL数组

- IDL数组运算简介

IDL面向矩阵的特性保证数组运算时不用进行循环。

IDL中使用数组的两个最大优势体现在：

(1)数组操作比循环操作快得多

(2)数组语法比相对的循环语法更加精练

例子：对数组元素求和

**Npts=1000000L**

**Data=randomu(-1L,npts)**

**Sum=total(data)**      IDL数组操作语句

**Sum=0.0**

**For i=0L,(npts-1L) do begin sum=sum+data[i]** 循环语句



- 数组的创建

(1)利用[]创建。

(2)利用创建函数创建

Byte     bytarr()   bindgen()

Int       intarr()    indgen()

long      lonarr()   lindgen()

ulong     ulonarr() ulindgen()

float      ftarr()    findgen()

double    dblarr()   dindgen()

string    strarr()   sindgen()

make\_array   make\_array(3,2,/byte)

- 数组的存储格式

**IDL**中数组元素的存储是按列进行的。按列存储的方式意味着连续的数组元素也将按顺序被存储，而且数组的第一维（列）变化的最快。

- 数组的下标

数组的下标可以是标量也可以是矢量。

进行下标操作时，如果下标超过了范围，则该下标被转换为在允许范围内的最小或最大的下标值。

## 1.3、表达式和运算法则

- 表达式的书写规则

表达式结果的类型将由表达式右边变量的类型决定。

- 运算符

数学: **+ - \* / ^ mod <取小 >最大**

逻辑: **Not Eq Ne Le Lt Ge Gt And Or Xor**

数组: **#数组乘(列乘行) ##矩阵乘(行乘列)**

指针: **\***

- 运算符的优先级

**()>指针>^>数组 / mod>+ - <> Not>逻辑**

## 1.4、数组运算

- 如果表达式中有一个变量是数组，结果也是数组
- 如果表达式左边是数组，右边是一个标量，则整个数组将被赋予该标量的值。

**A=[1,2,3,4] B=[3,4,5,6]**

**A+B, A\*B**

**A=10.0 B=[10.0,20.0,30.0,40.0]**

**A+B,A\*B**

- 取大(>)、取小(<)操作

分别返回自变量的最大值和最小值

当自变量为数组时，取大、取小运算符将对数组对应的每一对元素依次操作。

```
arr=[0,1,2,-9,5,6,-8,7,8] print,arr>0
```

```
a=[2,4,6,7] b=[4,5,5,6]
```

```
print, a<b
```



- 关系运算符

**eq ne le lt ge gt**

返回一个数字结果，其中真值用‘1B’表示，假用‘0B’表示。

关系运算符也可以作用于数组的自变量，如果两个自变量都是数组，那么关系运算符将依次作用于两个数组中每一对对应的元素。

```
a=1.0 b=2.0 help,a gt b
```

```
if (a lt b) then print,'True'
```

```
a=5 b=bindgen(9)
```

```
print,b
```

```
c=b le a
```

```
Print,c
```

```
a=[2,4,6,8] b=[3,4,5,6] print,a gt b
```

可以用于屏蔽数组中的某些值。

```
arr=indgen(9) mask=arr ge 5 result=mask*arr
```



## 1.5数组操作函数

- 数组元素的数目

**n\_elements**函数返回数组中所有元素的数目：

```
arr=findgen(32,32) print,n_elements(arr)
```

- 数组的大小和类型

**size**函数返回一个长整型的矢量结果，包含了输入数组的大小和类型信息。可选关键字

**n\_dimensions,dimensions,type,tname,n\_elements**分别返回维数、每维的大小、类型代码、类型名称和元素个数

```
arr=dist(256)
```

```
help,arr print,size(arr,/dimension)
```

```
print,size(arr,/type) print,size(arr,/tname)
```

```
print,size(arr,/n_elements)
```

- 最大值和最小值

**max min 函数**

```
arr=dist(32) print,min(arr),max(arr)
```

- 总和

**total**函数返回数组元素的总和，也可以添加可选变量来计算特定维度的元素总和。

**total**函数还可以使用可选的关键字**cumulative**来计算累计的和。

```
arr=indgen(3,3)
```

```
print,total(arr)
```

```
arr=indgen(9) print,total(arr,/cumulative)
```

## 1.6数组中元素的定位

- 查找符合条件的值

**where**函数返回数组或数组表达式中非零元素的下标，使用**where**函数和逻辑表达式可以查找符合条件的值。

**where**函数允许引入一个可选变量**count**来检查是否找到符合条件的值。

使用**where**函数时，数组使用一维下标

```
arr=indgen(9)*10
```

```
index=where(arr gt 35) print,arr[index]
```

## 1.7 改变数组的维度和大小

- 改变数组的大小  
    **rebin()** 通过整数因子数组的大小  
    **congrid()** 调整数组到任意大小

```
Arr=[20,30,40] resizeArr=rebin(arr,9,/sample)  
conArr=congrid(arr,9)  
conArr=congrid(arr,10)
```

## 二、编写IDL程序

### 2.1、定义和编译程序

- 过程(**pro**)

过程一般将几个相关的操作加到一个程序模块中。

过程以**pro** 开头，**end**结束

- 函数(**function**)

函数一般将一个操作加载到一个程序模块中，并返回结果

函数以**function**开头，**end**结束，并包括一个**return**语句返回结果

- 命名和编译源文件

命名一个**IDL**源文件的标志形式是在过程或函数名称后加上扩展名‘**.pro**’

过程和函数都可以通过‘**.compile**’命令进行手动编译或是通过**IDL IDE**环境进行编译。

当需要时，过程和函数会在运行时自动编译，如果**IDL**调用的过程或函数之前未被编译过，则**IDL**会搜索路径下所有的文件夹以搜索源文件的名称。

## 2.2控制语句

- If 语句  
if 条件 then 语句  
if 条件 then begin  
    语句  
endif  
if 条件 then 语句 else 语句  
if 条件 then begin  
    语句  
endif else begin  
    语句  
endelse



- **Case语句**

**case**语句根据一个标量的表达式来选择某个语句或语句块运行。

**case** 表达式 **of**

情况1:

情况2: 语句

情况3: **begin**

语句

**end**

**else:** 语句

**endcase**

当表达式和其中的某个情况匹配，相应的语句被执行， **case**语句结束，如果没有匹配的情况，那么执行**else**下的语句，如果没有**else**语句，将会发生错误，建议在**case**语句中都加上**else**

**test\_case.pro**



- **For语句**

**for** 语句每次循环执行一个语句或语句块

**for i=v1,v2 do** 语句

**for i=v1,v2 inc do** 语句

**for i=v1,v2 inc do begin**

语句

**endfor**

默认条件下增量为1，也可以设定增量

- **While语句**

当特定的条件为真，**while**语句执行单个语句或语句块

**while** 条件 **do** 语句

**while** 条件 **do begin**

语句

**endwhile**

Test\_for.pro

- **Return**语句

**return**语句在当前程序单元中产生一个即时出口，并返回控制。

**return**,结果

**return**

- **Break**语句

**break**语句用来中断循环的执行

**break**

- **Continue**语句

**continue**语句终止本次循环的执行，执行下次循环

**continue**

Test\_return.pro

Test\_break.pro

Test\_continue.pro

## 2.3 参数和关键字

- 参数

参数用来将变量和表达式传递到过程或函数中。参数有时也被称为位置参数，因为它们在自变量中的位置决定了它们在调用的过程或函数中如何被使用。参数通常作为输入/输出自变量，因为这些自变量在过程或函数中不可缺少。

- 关键字

关键字为可选自变量或表达式，它可以传递给调用程序，但不是强制性的。输入关键字可以用来指定一个自变量，或者用来设置一个布尔标识。

- 使用参数和关键字

在传递给过程或函数之前，输入的参数必须先定义。一个合理的过程或函数，都必须先检测其中任何的强制性输入自变量是否已经定义，然后在进行其他操作。

输出位置参数和关键字通常在过程或函数中创建，因此不需要在调用过程或函数时定义。

- 检测参数和关键字

**n\_params()** 返回传递的参数数目

**n\_elements()** 返回一个变量中元素的数目(零表示未定义)

**size()** 返回一个变量的类型和大小信息

**keyword\_set** 用于检测布尔关键字，如果自变量定义则返回为真

Eplot.pro

## 2.4全局变量

- 只读的系统变量

保存了当前**IDL**时间段的信息。

**!d.name** 当前图像设备的名称

**!d.window** 当前图形窗口的索引

**!d.table\_size** 当前颜色表的大小

- 可写的系统变量

可写的系统变量可以用来调整**IDL**的默认设置。如果用户改变了可写系统变量的值，那么这种改变在所有层次上都是有效的。

**!p.multi** 设置多面板成图

**!p.font** 为图形中的字符选择默认字体

**!order** 控制图像显示顺序

## 三、ENVI/IDL二次开发介绍

### 3.1 如何进行ENVI的扩展

- ENVI的扩展

**ENVI**是使用**IDL**语言编写的优秀的遥感影像处理平台。在**ENVI**中，用户可以很方便的通过**IDL**语言以及**ENVI**提供的二次开发工具对**ENVI**的功能进行增强，添加新的功能函数。

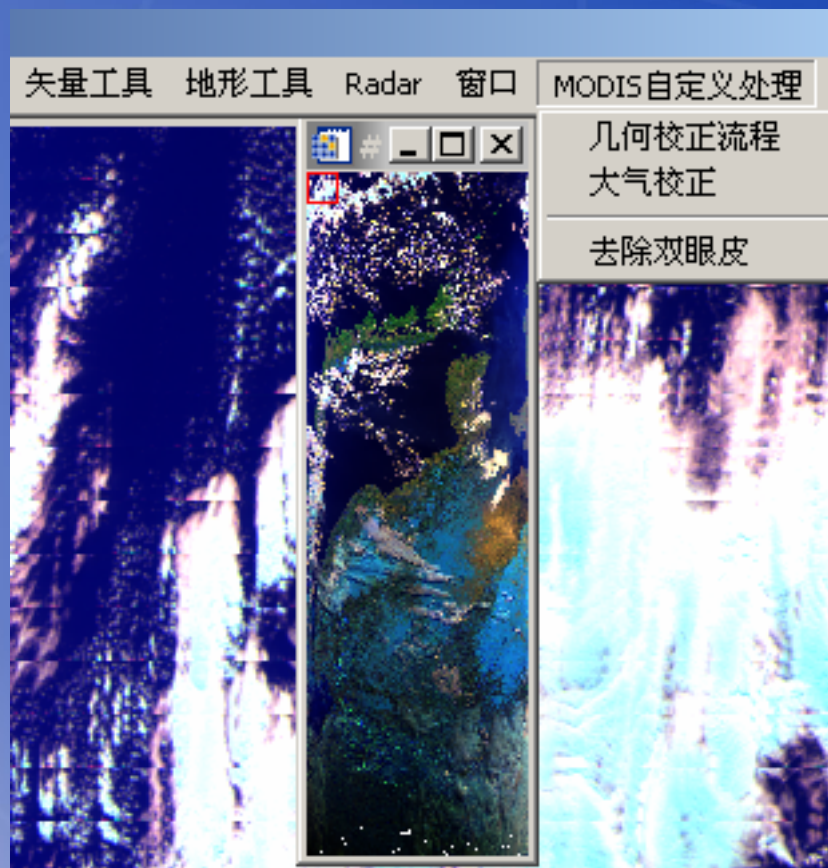
通常**ENVI**的扩展，包括波段和光谱算术函数，自定义的空间，光谱，或是感兴趣区域（**ROI**）的处理，用户函数，自定义文件输入程序，批处理，以及其它的报告和绘图工具。一系列**ENVI**程序能够为程序员使用，将能够极大地简化用户定制程序的开发，并保持和**ENVI**一样的外观。



- 波段和波谱运算函数

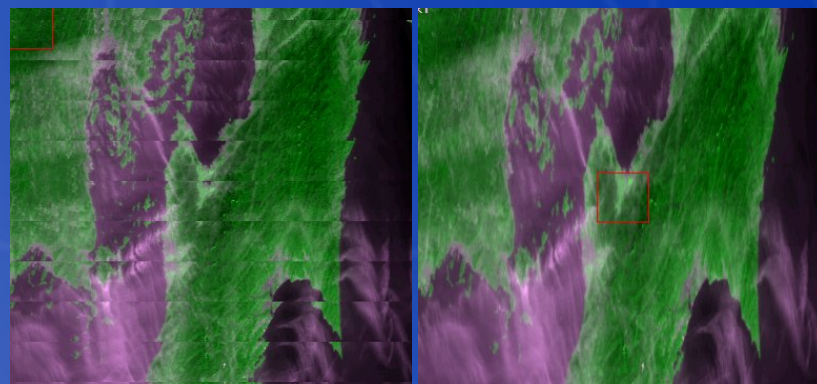
对于用户来说，扩展**ENVI**功能的最简单方法就是使用波段和波谱运算函数。波段运算和波谱运算是**ENVI**提供的两个功能，它们使得用户能够对波段或是波谱进行数学运算，或是使用用户编写的运算函数进行运算。波段和波谱运算函数，不需要处理文件I/O,不需要进行事件控制，不需修改菜单，用户只需编写运算函数部分内容，其它由**ENVI**进行管理。

# ENVI功能扩展



- 使用IDL编写代码，实现功能

- 将自定义功能添加到ENVI菜单中，与ENVI系统无缝集成



自定义去除双眼皮功能



- 用户函数

用户函数可以用**IDL**、**C**、**Fortran**或者其它的高级语言编写，并集成到**ENVI**软件中，通过**ENVI**的菜单来执行。用户函数可以通过**ENVI**获得输入数据，并将结果直接输入到**ENVI**中。

用户函数包括了部件的定义，事件的处理，以及处理程序。用户函数和**ENVI**菜单的一个按钮联系起来，并像**ENVI**的其它函数一样执行。

- **ENVI批处理模式**

在批处理模式下， **ENVI**的非交互函数，可以通过用**ENVI\_DOIT**函数来实现。**ENVI\_DOIT**函数提供了处理部分，而不需用户交互就能够运行。批处理模式可以通过菜单事件或是**ENVI**非交互模式来开始。

## 3.2 ENVI下编写程序的特点

- 无用户交互下的复杂程序的控制

传统的**ENVI**处理程序需要大量的用户交互来获取处理所必需的信息。因此在编写无用户交互程序时，用户必须确定**ENVI**处理函数所必须的参数，**ENVI**函数要比传统的**IDL**程序拥有更多的参数。

## ENVI下的文件I/O和IDL下I/O区别

- 在IDL中，文件I/O需要获得该文件的逻辑单元号（LUN），并使用文件读写程序如，**OPENR**、**READU**、**OPENW**以及**WRITEU**等对文件进行读写。
- 在ENVI程序中的所有文件I/O都是由ENVI进行内部控制的，因此ENVI的程序员就没有必要获取LUN。相反，所有需要输入文件的ENVI程序要确定一个特定的文件ID，**FID**。**FID**基本上类似数据文件的指针，但它不是LUN。当一个文件需要被访问，ENVI内部获得该文件的LUN，读写需求的数据，然后释放LUN。这样ENVI无需使用或保存任何LUN。
- 与使用**OPENR**打开文件相反，ENVI提供了几种不同的程序函数来打开文件。当文件被打开时，这些函数都会返回一个文件**FID**。

### 3.3 ENVI程序中的通用关键字

- **FID**

**FID**是一个长整型的标量。**FID**为**ENVI**的程序员提供了一个命名变量，可以用于一个或几个**ENVI**程序，来打开或选择文件。所有对该文件进行操作的**ENVI**程序都是通过**FID**完成。

需要注意的是，**FID**和**LUN**是不同的。如果文件打开失败，则**FID**返回为-1

- **R\_FID**和**M\_FID**

**ENVI**处理程序产生结果一幅新图像也包括一个**R\_FID**,或者称为返回**FID**关键字。如果结果是存在内存中的，设置**R\_FID**关键字是访问数据的唯一方法。

运行进行掩模的处理程序还包括一个**M\_FID**，或者称为掩模关键字用于确定用作掩模波段的文件。

- **DIMS**

**DIMS**关键字是一个5个元素长整型数组。它定义了处理数据的空间子集。当需要确定**FID**的时候，你必须同时确定该文件的空间范围。

**DIMS[0]** 存储一个打开的**ROI**区域的指针，仅在**ROI**被定义的时候使用，其它时候设为-1L

**DIMS[1]** 采样的起始位置 **Sample start**

**DIMS[2]** 采样的终止位置 **Sample end**

**DIMS[3]** 行的起始位置 **Line start**

**DIMS[4]** 行的结束位置 **Line end**

- **POS**

**POS**关键字定义了用于处理的波段位置，是一个变长的长整型数组。波段从0开始，例如，要处理第三波段和第四波段，**POS=[2,3]**



## 3.4常用ENVI功能函数介绍

- 文件管理

**ENVI\_PICKFILE:** 产生一个提示用户选择文件的对话框，并返回用户所选择的文件名

**ENVI\_SELECT:** 产生对话框提示用户从**ENVI**中已经打开的文件中选择一个文件，并返回用户所选择文件的**FID**，该函数还可以返回**DIMS**和**POS**的值

**ENVI\_OPEN\_FILE:** 该函数返回一个文件的**FID**，并将文件信息添加到可用波段列表中

**ENVI\_FILE\_MNG:** 该函数可以打开、关闭或者删除硬盘上的文件。无需用户干预

**ENVI\_GET\_FILE\_IDS:** 该函数返回所有当前打开的文件的**FID**

- 打开外部文件格式

**ENVI\_OPEN\_DATA\_FILE:** 该函数打开**ENVI**所支持的外部文件（无**ENVI**头文件）并返回**FID**

- 获取数据

**ENVI\_GET\_DATA:** 该函数从一个打开的文件中获取影像数据。它每次只返回某一波段的数据，数据的范围由**DIMS**关键字控制。

**ENVI\_GET\_SLICE:** 该函数从一个打开的文件中获取波谱影像数据，它返回影像某一行所有波段的数据值。结果以**BIP**或**BIL**的格式返回



- 将数据输入到内存

**ENVI\_ENTER\_DATA:** 该函数将IDL数组中的数据输入到内存中，并通过可用波段列表进行管理。

- 将影像数据存入硬盘

使用IDL的WRITEU函数写入数据

**ENVI\_SETUP\_HEAD:**使用该函数写某个影像数据的头文件

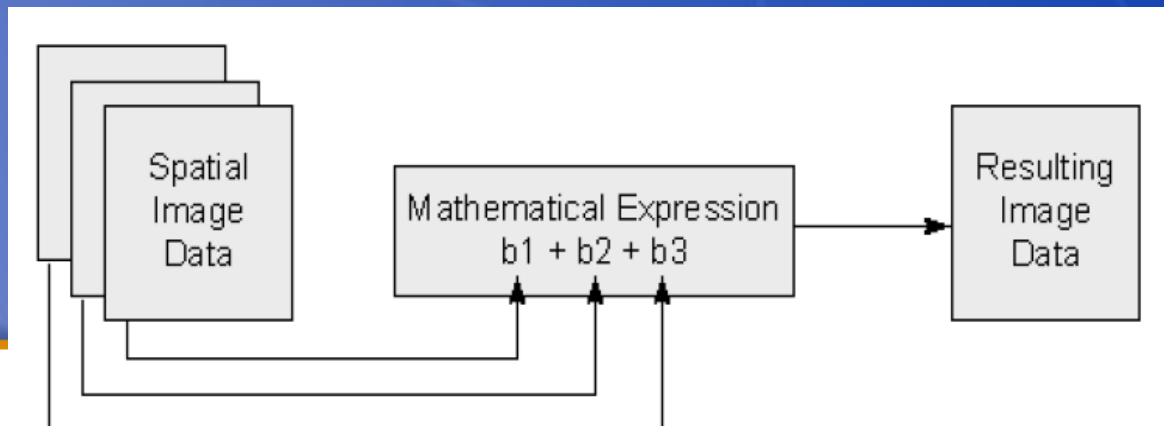
## 四、波段和波谱运算函数

### 4.1 波段运算

- 波段运算基础

**ENVI**波段运算函数能够调用用户编写的程序进行定制的处理。波段运算接口用来定义波段或文件作为输入，调用用户编写的函数，并将结果写入文件或是输入到内存。

波段运算函数使用变量命名为**b1(B1)**,**b2** 等等。通过在波段运算表达式窗口中输入函数名和变量名就可以调用波段运算函数。变量值通过波段运算对话框选定。



- 编写波段运算函数

波段运算函数在波段表达式中执行，它的编写非常简单。该函数接受输入波段，处理数据，并返回结果。函数以下面的方式定义：

**Function bm\_func,b1,[b2,...,bn, parameters and keywords]**

**processing steps**

**return,result**

**end**

- 编译波段函数

一旦函数完成后，结果**.pro**或**.sav**文件需要放到**ENVI**安装目录下的**save\_add**目录下，这样，每次**ENVI**启动时就会自动编译或恢复该函数。当然，也可以通过在通过选择**File—Compile IDL Module**进行编译。

注意：**ENVI RT**不能**.pro**文件，只能接受**.sav**文件。

- 波段函数的例子

下面的例子创建一个自定义的波段运算函数执行下面的运算并检查是否能被0除。

$$(b1+b2) / (b1-b2)$$

**bm\_ratio.pro**

## 4.2 波谱运算

- 波谱运算基础

波谱运算函数使用变量名为**s1(S1)**,**s2**等等。波谱运算函数在波谱运算表达式中被调用。通过波谱运算对话框确定变量。

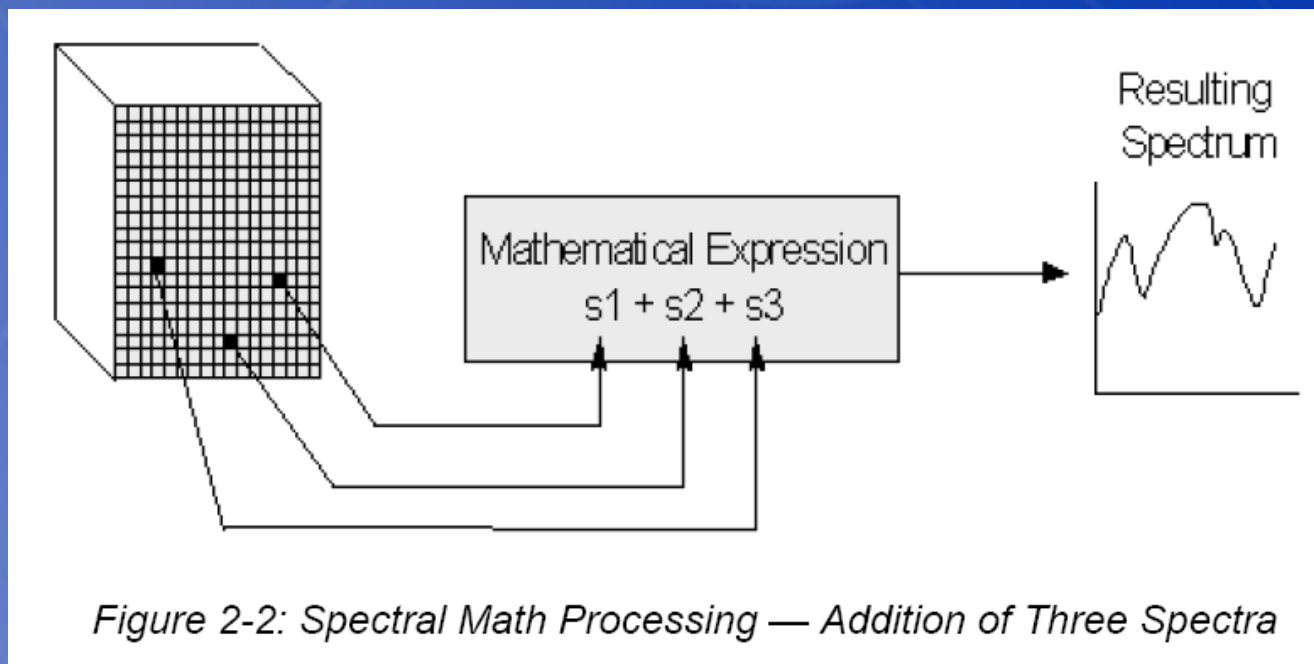


Figure 2-2: Spectral Math Processing — Addition of Three Spectra

- 波谱运算函数格式

```
function sm_func, s1, [s2,..., sn, parameters and keywords]  
    processing steps  
    return, result  
end
```

- 波谱运算的例子

要执行如下的运算: **s1 / s2**

**sm\_ratio.pro**

## 五、ENVI批处理模式

### 5.1 批处理模式简介

- 以批处理模式运行**ENVI**能够让用户在命令行模式下使用**ENVI**。这种能力在以下几种情况下非常有用：
  - （1）用户主要使用**IDL**工作但需要偶尔用到**ENVI**的函数；
  - （2）用户希望创建定制的应用程序混合了**IDL**代码和**ENVI**函数；
  - （3）用户希望进行大量的**ENVI**处理而无需人工干预。



- 批处理模式的**ENVI**和正常模式下没有什么区别，只是通过一系列的特定的函数库来执行**ENVI**的功能。为了使用这些函数，必须首先将它们恢复到**IDL**内存中。因此为了正确获取这些函数，有必要了解一下**ENVI**程序的结构。
- **ENVI**功能文件由大约**40**个小的**IDL save**文件组成，这些文件是包括数据和编译后的程序的二进制文件。这些**save**文件存放在**ENVI**安装目录下的**Save**目录下。**ENVI**的核心**save**文件包括**ENVI**的基本功能函数，动态运行函数以及**ENVI**运行所需的内部变量。

## 5.2如何开始批处理模式

- 恢复ENVI sav文件  
`envi, /restore_base_save_files`
- 开始批处理模式  
`envi_batch_init, log_file='batch.txt'`
- 退出批处理模式  
`envi_batch_exit`

`bt_init.pro`

## 5.3 批处理程序例子

查看DEM **view\_dem.pro**

文件信息统计 **bstats1.pro**

饱和度拉伸 **stastrch.pro**

## 六、用户函数

### 6.1 用户函数介绍

- 用户函数允许用户为**ENVI**添加新的功能并通过**ENVI**的菜单进行访问。用户能够添加任意数量的用户函数，并且每个函数都可以获得它自己的菜单选项。当用户通过菜单选择用户函数时，将会执行这些函数，如同**ENVI**的其它函数一样。用户函数和**IDL**程序没有什么区别，可以称为**ENVI**的程序。在某种意义上说，它们和批处理模式也很接近，除了无需初始化批处理模式。

- 用户函数是事件的处理程序。因此，所有的**ENVI**用户函数必须遵循事件处理的基本规则，即用户函数定义时必须加上一个附加的变量来接受事件结构。即使用户函数中不会用到事件信息，这个附加参数也必须加上。

**Pro ProName, ev**

**Function FunName,ev**

## 6.2修改ENVI的菜单

- ENVI菜单系统介绍

ENVI的菜单系统，包括主菜单和显示窗口菜单，是通过在ENVI安装目录下MENU目录下的envi.men和display.men这两个ASCII码文件中进行定义的。

MENU目录位置

Windows系统上:

X:\ris\idlxx\products\envixx\menu

UNIX和Mac OS X系统上

/usr/local/rsi/idl\_x.x/products/envi\_x.x/menu

Envi.men文件定义了ENVI的主菜单，display.men文件定义了显示窗口菜单。每次ENVI启动的时候，这两个文件被读入并根据它们的内容构建ENVI的菜单。要在菜单中添加内容，只需在这两个文件中添加一行，并重新启动ENVI。

- **ENVI菜单系统结构**

使用任何文本编辑器就可以打开**envi.men**文件。文件的结构如下所示：

**0 {File}**

**1 {Open Image File}{open envi file}{envi\_menu\_event}**

**1 {Open Vector File}{open vector file}{envi\_menu\_event}**

**1 {Open External File}**

**2 {Landsat}**

**3 {Fast} {open fast tm} {envi\_menu\_event}**

**3 {GeoTIFF} {open tiff} {envi\_menu\_event}**

**3 {HDF} {open envi file} {envi\_menu\_event}**

**3 {NLAPS} {open nlaps} {envi\_menu\_event}**

每一行开始的数据定义了菜单项的层次。**0**表示最顶层，**1**表示一级子菜单，**2**表示二级子菜单，等。



- **{Open External File}**第一个大括号括起来的部分定义了显示在菜单上的内容。
- **{open envi file}**第二个大括号括起来的部分定义了为菜单项所赋给的用户值。用户值在同一用户函数处理多个菜单项时非常有用，可以区别那个菜单项被选择。
- **{envi\_menu\_event}**第三个定义了菜单项事件处理程序的名称，即编写的用户函数名。此处使用的是用户函数名，而不是用户函数所在的文件名，所以没有后缀。
- 需要注意的部分：用户值在大多数**ENVI**的程序中是需要的，要保持用户值的唯一性。但当编写用户函数时，大多数情况下，用户值部分是没有用的，这时候，可以将用户值设为和用户函数名一致，也可以将它设置为**{not used}**等醒目的标示。

## 6.3编写一个用户函数的实例

- 修改ENVI菜单

添加如下部分：

**0{MyFunctions}**

**1{Basic File Info}{not used}{file\_info}**

保存菜单文件

- 编写用户函数

**file\_info.pro**

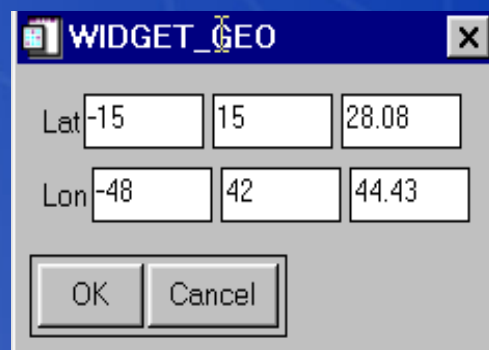
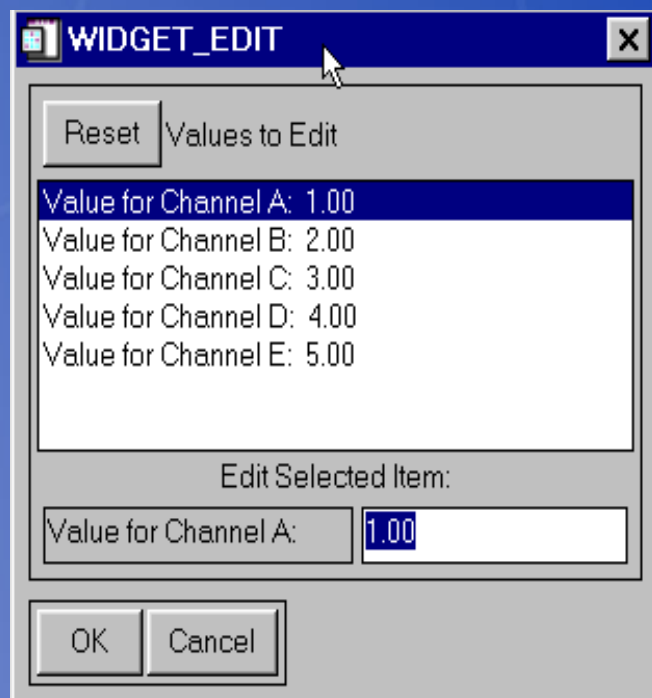
## 6.4为用户函数添加小部件

- 使用小部件收集用户输入比通常的IDL程序中使用**Widget**要容易，因为用户无需编写事件处理程序，**ENVI**自动管理事件。此外**ENVI**的复合部件包括在**ENVI**的函数中，这样就能够很方便的创建和**ENVI**类似的界面。

## 6.5可用的ENVI小部件

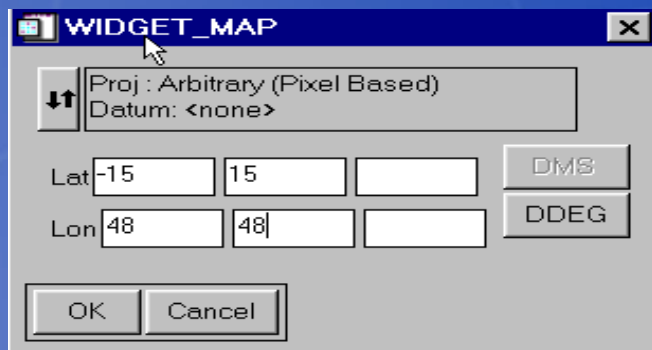
- **ENVI**包括了**20**多种的小部件，可以为用户函数所用。大多数的函数以**WIDGET\_**开头。
- **ENVI\_PICKFILE**: 用于从硬盘上选择一个文件。可以用来收集任意类型的文件名。
- **ENVI\_SELECT**: **ENVI**标准的文件选择对话框，用来选择一个打开的文件，确定空间和光谱子区，以及掩模波段。它也包括了一个打开按钮，能够允许用户从硬盘上打开一个新的文件。
- **WIDGET\_EDIT**: 提供了一个部件从列表中选择项目。

- **WIDGET\_EDIT:** 提供了一个部件从列表中选择项目



- **WIDGET\_GEO:**用于提示用户选择经纬度值。

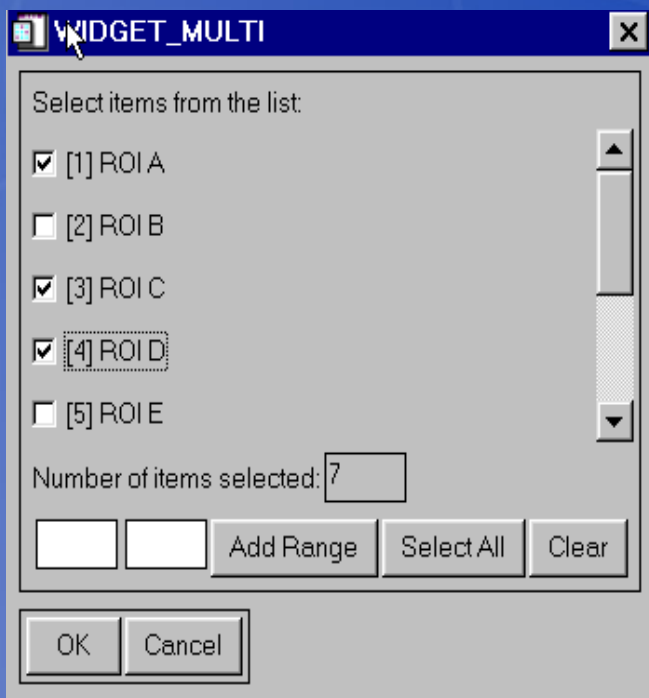
- **WIDGET\_MAP:**用于编辑地图坐标和投影



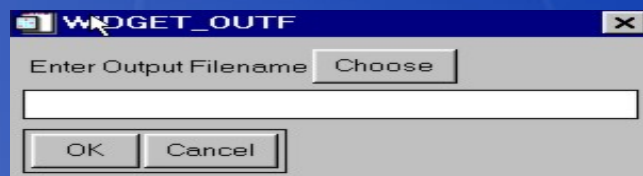
- **WIDGET\_MENU**



- **WIDGET\_MULTI**:用于多项选择

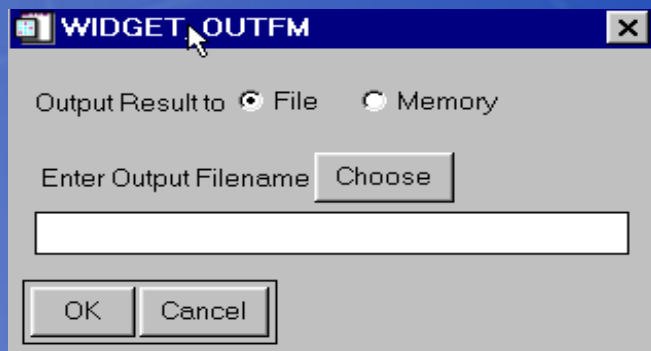


- **WIDGET\_OUTF**  
用于选择一个输出文件名

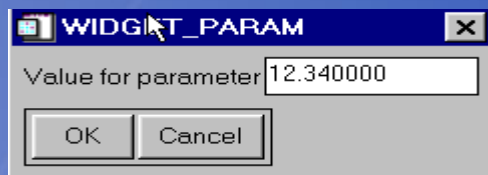




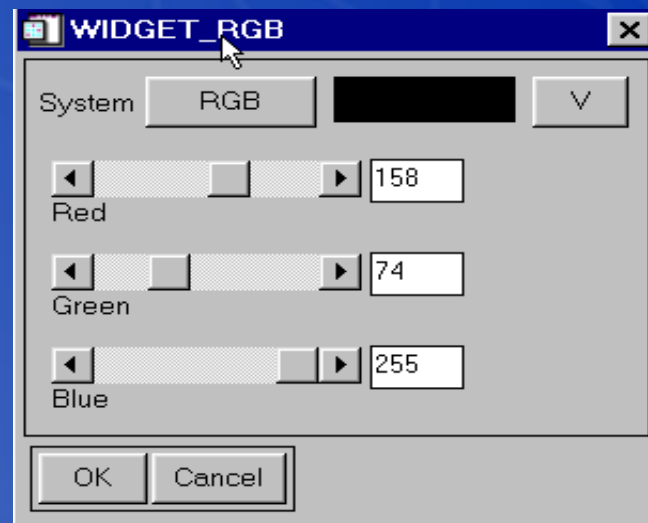
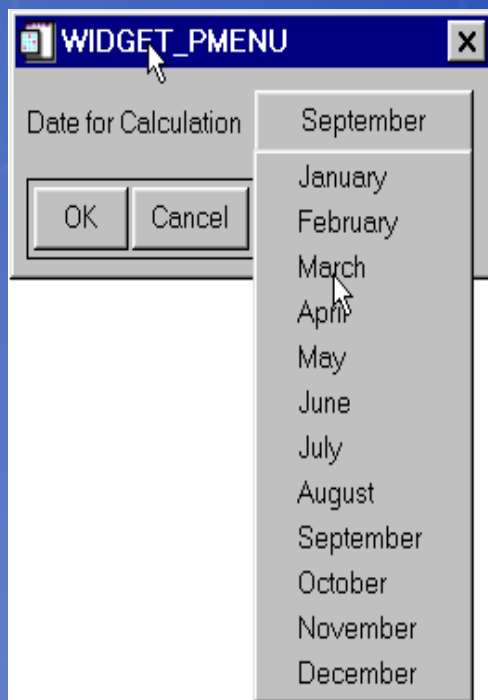
- **WIDGET\_OUTFM:**用于选择一个输出文件名或是输入到内存



- **WIDGET\_PARAM:**

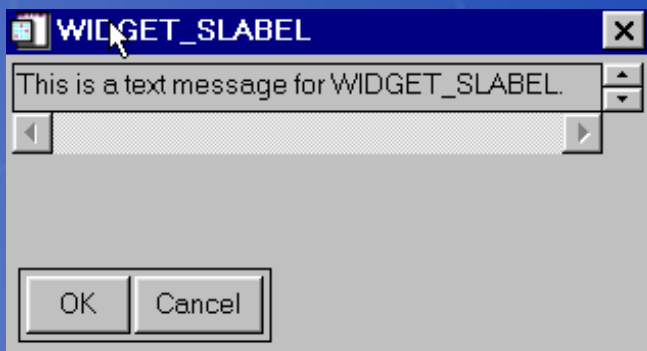


- **WIDGET\_PMENU:**提供下拉菜单

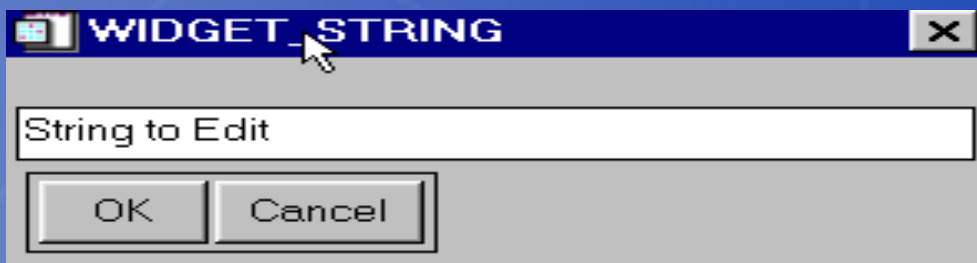


- **WIDGET\_RGB:**用于修改RGB颜色值

- **WIDGET\_SLABEL:**用于显示文本信息



- **WIDGET\_STRING**



## 6.6小部件事件自动管理程序

- 通常在**IDL**程序中，程序员要对程序中使用的小部件编写事件处理程序。这对新使用**IDL**的用户来说，是一个比较大的难题。因此为了方便使用**ENVI**用户函数，**ENVI**中提供了自动事件处理程序。这种自动事件处理程序能够自动的管理所有**ENVI**部件产生的事件。主要有两个函数进行事件的自动管理。
- **WIDGET\_AUTO\_BASE**  
在通常的**IDL**程序中，所有的**BASE**部件，包括顶级**BASE**，都是通过**WIDGET\_BASE**函数创建的。但是，在**ENVI**编程中，如果要创建事件自动管理的部件构架，必须通过**WIDGET\_AUTO\_BASE**创建顶级**BASE**，在构建**GUI**的过程中使用的其它**BASE**使用原来的**WIDGET\_BASE**函数创建。使用**WIDGET\_AUTO\_BASE**函数创建的顶级**BASE**是自动列对齐，居中和模态化的。**WIDGET\_AUTO\_BASE**仅介绍几个关键字来控制它的属性。

- **AUTO\_WID\_MNG**

在通常的IDL部件程序中，一旦**GUI**被定义，**XMANAGER**程序会被调用进行部件的注册并进行部件事件的检测。而在自动事件管理的**ENVI**程序中，无须调用**XMANAGER**程序，相反，一个**ENVI**函数**AUTO\_WID\_MNG**被调用进行部件的注册，检测事件，并以结构的形式返回用户输入的值。

小部件事件自动管理的例子：**test\_widgets.pro**

## 6.8 用户函数中错误捕获

### 防止错误的简单方法

- 当一个部件包括**Cancel**按钮，一定要检查**Cancel**按钮是否被选择。这种错误检测非常容易实现，应该总是包括在用户函数中。
- 如果可能，在代码中应设定变量的默认值，以防止用户忘记输入参数值。
- 当使用**WHERE**函数时，一定要使用**COUNT**参数来确保合法的结果被返回。

## I/O错误处理

- 通过IDL函数**ON\_IOERROR**来完成。当I/O错误发生的时候，**ON\_IOERROR**程序能捕获该错误并跳转到错误处理程序。当错误被捕获时，错误状态存储在系统变量**!ERROR\_STATE**中错误信息存储在**!ERROR\_STATE.MSG**变量中。
- **CATCH**函数，它提供了一种更通用的机制来处理异常和错误。**CATCH**函数的优点在于它不仅能够捕获I/O错误，而且能够捕获任何程序错误，包括未定义变量、非法的数组下标或是未定义的函数。
- **ENVI**提供了**ENVI\_IO\_ERROR**函数来显示错误信息，使用这个函数使用用户函数的错误显示和**ENVI**系统的错误显示保持一致。



## I/O 错误处理 实例

本例说明了使用**ON\_IOERROR**和**ENVI\_IO\_ERROR**进行错误捕获和显示处理的过程。强烈推荐所有的处理程序都进行I/O错误的处理和显示相关的错误信息。下面的步骤列出了在处理程序中处理I/O错误的步骤：

- (1)在程序开始，清除系统错误代码，并定义I/O错误跳转语句。
- (2)在程序结束时，清除系统错误代码
- (3)判断是否产生I/O错误并打印出错误信息
- (4)删除当前的输出文件

```
pro user_function, [parameters and keywords]  
message,/reset  
on_ioerror, trouble  
Initialization and Processing ...  
message,/reset  
trouble: if (!error_state.code ne 0) then $  
envi_io_error, 'User Function', unit=unit  
if (!error_state.code eq 0) then $  
Write an ENVI header  
End
```

- 使用**Catch**函数进错误的捕获

**Catch, error**

**IF (error NE 0) THEN BEGIN**

**ok = DIALOG\_MESSAGE(!error\_state.msg, /cancel)**

**IF (STRUPCASE(ok) EQ 'CANCEL') THEN return**

**ENDIF**

## 6.9和显示窗口进行交互

- **ENVI**中每一个三窗口的显示组都能够通过一个唯一数字标识**DN**进行区别，**DN**以**0**开始。
- **ENVI\_DISP\_QUERY**: 该函数体能够了获取当前显示影像的基本信息，包括影像文件的**FID**，空间分辨率，影像的显示类型（**RGB**，灰度或分类），显示的波段位置，以及三个窗口的大小。
- **ENVI\_GET\_IMAGE**: 该函数类似于**ENVI\_GET\_DATA**函数，但它用于从显示窗口中返回数据。给定波段位置，维度，以及**DN**值，**ENVI\_GET\_IMAGE**函数能够返回拉伸后的灰度值。



- **DISP\_GET\_LOCATION**

该函数返回当前选定的像素的位置。

- **DISP\_GOTO**

该函数移动**Zoom**窗口到一个指定的位置，并在必要的情况下更新**Image**和**Scroll**窗口。

## 6.10使用影像分块技术

### 影像分块技术简介

- **ENVI**的处理函数获取输入影像数据，处理数据，并输出新的影像，绘图或是提供报告。**ENVI**的处理函数通常都是和**ENVI**的影像分块技术集合在一起，以处理任意空间和波谱大小的影像。
- 空间分块的大小能够在配置文件中定义，而波谱分块的大小总是等于采样的数目乘上波段数(**sample\*band**)。

- 所有的**ENVI**用户函数也能够通过**ENVI**内建的分块函数获取数据。这确保了用户函数也能够处理任意大小的数据文件。**ENVI**的分块来自于三种格式：**BSQ**格式，**BIL**格式以及**BIP**格式。**ENVI**还提供了进度条部件来显示分块的处理情况。
- **ENVI**也提供了未使用分块技术的函数，但是不推荐使用，因为它仅能用于比较小的文件。当然未使用分块技术的函数是一种快速的访问数据的方法，因此可用于快速进行程序原型的开发。



- 分块处理程序

**ENVI**分块处理将输入数据分成同样大小的单元，可以是空间方式也可以是波谱方式，以确保所有大小的影像都能被处理。一个空间分块的大小是 $n$ 行\*所有列，而波谱分块的大小是 $\text{Sample} * \text{band}$ 。

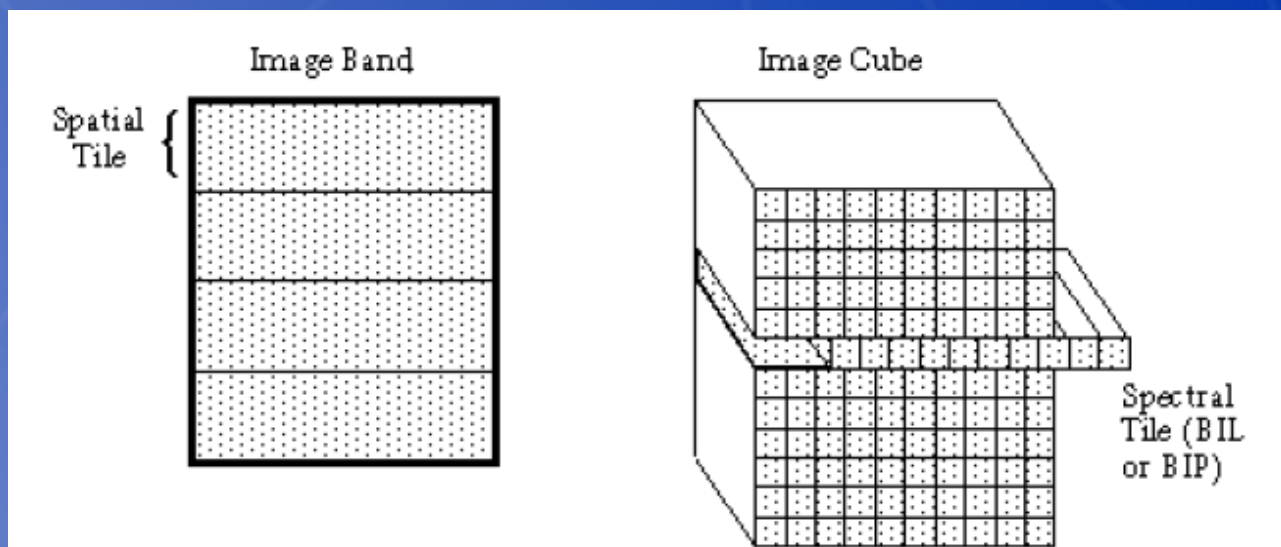


Figure 4-19: Illustration of Spatial and Spectral Tiles

- 空间分块近似等于按输入波段对影像进行分块，因此可以不用考虑文件的存储方式而进行空间处理。但访问单一文件的多个波段，所有波段将拥有同样数目的空间分块。通常进行空间分块的都是**BSQ**文件，而进行波谱分块的是**BIL**或**BIP**文件。使用和输入文件同样的存储方式非常有效。
- 当进行邻域处理时，空间分块也可以设定重叠的行数。重叠行仅加在每个分块的顶部，在整个波段作为一个分块时，没有重叠行。例如：进行**3x3**卷积时，需要一行重叠来处理上一个分块的最后一行。

- 分块处理的步骤如下：  
初始化空间或波谱分块需求，使用**ENVI\_INIT\_TILE**  
获取分块输入数据，**ENVI\_GET\_TILE**  
当所有的分块数据都处理完毕，释放分块需求，**ENVI\_TILE\_DONE**
- 空间分块的例子 **spat\_tile.pro**
- 波谱分块的例子 **spec\_tile.pro**
- 根据输入文件存储方式自动确定空间分块还是波谱分块  
**ss\_tile.pro**

- 保存结果

输出文件通过使用IDL程序**OPENW**写入，在调用**OPENW**程序前，需要通过**GET\_LUN**函数获得文件单元号。通过IDL程序**WRITEU**函数将处理后的分块数据写入文件。在所有分块数据都写入后，文件被关闭，文件单元号通过IDL程序**FREE\_LUN**释放。

一旦文件被写入硬盘，可以使用**ENVI**函数**ENVI\_SETUP\_HEAD**进行**ENVI**头文件的写入。下列文件信息必须写入头文件：文件名，采样数，行数，波段数，偏移，存储方式，以及数据类型。此外还有一些可选的关键字。如**X**、**Y**的起始位置，文本描述，波段名称等等。

- 对于内存输出，结果存储在内存中分配的数组中。处理后的数据块插入合适的存储位置。内存数组的大小为 $NS*NL*NB$ ，IDL函数BYTARR，INTARR，LONARR，FLTARR，DBLARR，以及MAKE\_ARRAY用来创建相对应的比特类型、整型、长整型、浮点、双精度浮点以及任意类型的内存数组。
- 当处理结果完成后，包含处理结果的内存数组可以使用ENVI\_ENTER\_DATA传递给ENVI。在最简单的情况下，仅仅内存数组是必须的。同样有一些额外的信息可以提供，如XY的起始位置以及文字描述和波段名称。

- 实例：空间分块结果存入硬盘
- **spat\_disk.pro**
- 实例：将分块处理结果存入内存
- **spat\_mem.pro**



## 6.11 非分块处理程序

- **ENVI\_GET\_DATA**

该函数从文件中获取数据，一次只能对单一波段操作，范围由**DIMS**关键字指定。该函数提供了**xfactor**和**yfactor**两个参数能够产生放大和缩小的影像。

- **ENVI\_GET\_SLICE**

该函数从文件中获取波谱数据，以**BIP**或**BIL**的格式返回。



## 6.12处理进度报告

- 处理进度报告显示了当前处理的完成程度。使用**ENVI**提供的处理进度报告，开发人员只需控制增量大小和更新频率。可选的关键字**Cancel**用来在下次增量更新时终止处理进程。处理进度报告由三个程序控制，分别为初始化、设置增量、更新状态。这些函数列在下面：

**ENVI\_REPORT\_INC**

设置报告的增量

**ENVI\_REPORT\_INIT**

初始化报告对话框

**ENVI\_REPORT\_STAT**

更新完成的百分数并

检查用户是否执行了**Cancel**

注：只有在处理进程小于**100%**时，用户才可以取消处理，如果处理进度已达到**100%**，**Cancel**将被忽略。



- 赋给**ENVI\_REPORT\_INIT**的字符串将以单独一行的方式显示在进度对话框上。通常**ENVI**使用两个字符串来区分输入和输出文件。为了保持一致性，输入的字符串数组也应如下设置。

**['Input File:filename','Output File: filename']**

- 对于输入内存的情况：

**['Input File : filename ', 'Output to Memory']**

- 通常进度对话框在分块循环内部进行更新，循环的总数为分块的总数。在这种情况下，进度报告的增量总量就设定为影像分块的总数。例如：对于**5**个分块，报告增量总数就设置为**5**，每次实现**20%**的递增。

## 例子：处理进度对话框

- 启动**ENVI**
- 在**IDL**命令行中键入以下命令：
- 初始化进度对话框：  
**Envi\_report\_init,['Input File:filename'],\$  
    'Output File:filename'],title='Test Status',base=base**
- 设置进度增量  
**ENVI\_REPORT\_INC,base,3**
- 进行进度更新：  
**ENVI\_REPORT\_STAT,base,1,3**  
**ENVI\_REPORT\_STAT,base,2,3**  
**ENVI\_REPORT\_STAT,base,3,3**
- 完成进度对话框  
**ENVI\_REPORT\_INIT,base=base,/finish**

## 6.13 编译用户函数

- 由于IDL编译器不能识别ENVI库函数，因此用户程序在编译的时候通常会报错。
- 同时为了向下兼容，IDL编译器将（）作为数组的定义，当IDL编译器不能识别函数时，它会将它当作是数组定义，从而导致编译错误。
- **FORWARD\_FUNCTION**可以告诉编译器，哪些变量是函数，而非数组定义。
- **COMPILE\_OPT STRICTARR**则强制编译器以[]作为数组的定义。

## RESOLVE\_ALL

- 在IDL程序中，用到许多IDL内置的函数，都是以源码的形式提供的。在IDL编译器中，它们被自动编译。但是在ENVI中，ENVI不能编译这些函数，因此要想将用户函数打包，必须要找到所有依赖的函数，而IDL提供了一个工具函数就是RESOLVE\_ALL，该函数可以自动寻找和编译用户程序所依赖的所有函数。
- 在使用RESOLVE\_ALL函数时要注意，它也不能识别ENVI库函数，在遇到ENVI库函数时会报错，因此在使用时，必须加上CONTINUE\_ON\_ERROR关键字。

## 例子编译用户函数

- 在用户函数中加入**COMPILE\_OPT STRICTARR**或是使用**FORWARD\_FUNCTION**函数
- 保存修改后的代码
- 启动一个新的**Session**
- 编译修改后的用户函数
- 使用**RESOLVE\_ALL**函数编译所有依赖函数：
- **RESOLVE\_ALL, /CONTINUE\_ON\_ERROR**
- 使用**SAVE**函数将用户函数存为**SAVE**文件
- **SAVE, file='my\_user\_function.sav', /routines**
- 注：**save**文件名必须和用户函数名一致，并包括一个**.sav**后缀，编译后的**SAVE**文件必须放入**ENVI**安装目录下的**save\_add**目录中。



## 七、ENVI提供的交互工具

### 7.1 绘图工具

- **ENVI**为用户提供了访问**ENVI**绘图窗口的方法。用户能够定义一系列的绘图并将其载入到绘图窗口中。**ENVI**函数**ENIV\_PLOT\_DATA**用于绘制**X**、**Y**数据。可选的关键字有绘图的标题，颜色，名称，线形，轴名以及各个绘图的标题。
- 例子： 绘制数据



## 7.2报告

- 用户可以使用**ENVI**提供的报告部件来显示文本数据。这个报告部件通过**ENVI\_INFO\_WID**创建，它能够将用户提供的字符串数组中的每一个元素以新的一行显示。**ENVI**将会自动管理报告部件的事件，用户无需编写事件处理程序。
- 例子：创建一个报告

## 7.3 RGB颜色三元组

- 在很多**ENVI**函数中要使用到颜色索引，而其它的函数使用**RGB**颜色三元组。**ENVI**函数**ENVI\_GET\_RGB\_TRIPLETS** 可以返回任何颜色索引的**RGB**值。为了避免颜色索引超出系统的颜色数，函数内部采用了求摸得运算。
- 注：用户可以添加自定义的颜色到**ENVI**系统中，通过修改**ENVI** 目录下的**colors.txt**文件。
- 例子：获取**RGB**颜色值

## 7.3 获取文件信息

- ENVI提供了**ENVI\_FILE\_QUERY**函数用来获取文件的信息，这些文件信息可以用于**ENVI\_SETUP\_HEAD**和**ENVI\_ENTER\_DATA**等函数。
- 例子：获取文件的基本信息
- 例子：获取地图信息

## 7.4 文件管理工具

- **ENVI**提供了几个函数可以进行文件的管理：
- **ENVI\_FILE\_MNG** 管理打开的文件，可以将打开的文件关闭，删除
- **ENIV\_OPEN\_DATA\_FILE** 打开一个**ENVI**支持的外部文件
- **ENVI\_OPEN\_FILE** 打开一个影像文件
- **ENVI\_OUTPUT\_TO\_EXTERNAL\_FORMAT** 将**ENVI**文件输出为外部格式
- **ENVI\_PICKFILE** **ENVI**的文件选取函数
- **ENVI\_SELECT** 选择一个打开的**ENVI**图像文件

## 综合实例

- **Reset\_of\_image\_roi.pro**
- 本例是一个**ENVI**用户函数，需要在交互式**ENVI**模式下执行，它的功能是对当前显示的影像所定义的**ROI**区域进行求反，生成一个新的**ROI**区域。
- **[www.rsinc.com](http://www.rsinc.com)**