ENVI 二次开发指南

航天星图（北京）有限公司

2006 年 7 月

前　言

ENVI（The Environment for Visualizing Images）遥感影像处理软件是美国 RSI 公司推出的由专业的遥感科学家基于交互式数据语言 IDL 开发的一套功能强大的遥感影像处理系统，它可以轻松读取、显示、分析各种类型遥感数据，并提供了从影像预处理、信息提取到与地理信息系统整合过程中需要的各种工具。

ENVI 软件进行入中国市场十年来，凭借着其强大的遥感影像处理功能、丰富的遥感数据格式支持、简单易用的使用风格、中文化的菜单支持、全面的操作系统支持及 IDL 底层开发平台强大的可扩展能力被广大的遥感用户逐步熟悉和使用，并被广泛地应用于国土、地质、环境、林业、农业、军事、自然资源勘探和海洋资源管理等多个领域。

航天星图（北京）有限公司作为美国 RSI 公司在中国地区的独家代理商和增值服务商，不仅负责向广大用户提供 RSI 所有产品的销售和技术持，并且在 RSI 的全力支持下，为用户提供包括遥感数据生产流程设计、遥感影像增值服务、遥感平台定制开发、产品化设计和开发等全方位的技术服务，为用户的空间信息应用提供专业快速的解决方案。

面对越来越多的用户对 ENVI/IDL 二次开发中文手册的需要，我们特组织编写了 ENVI 二次开发指南。全书包括 ENVI 编程介绍、波段运算、批处理、用户函数、常用编程工具、交互式函数等 ENVI 编程的各个方面。书中附有大量的编程示例，详细说明了如何使用 ENVI 进行二次开发。

我们的联系方式：

E_mail: support@imagetekinfo.com

鉴于水平与时间有限，培训教程中不妥乃至错误之处在所难免，恳望学员不吝批评指正。

目 录

第一章 概述

一、ENVI 扩展简介

1、扩展 ENVI 是有可能的

扩展 ENV 包括一系列的定制。它包括创建波段和波谱数学函数、进行复杂图像处理、图形接口程序的开发。

通常的 ENVI 扩展，包括波段和波谱运算函数，自定义的空间、波谱，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，以及其它如报告和绘图工具等。ENVI 提供了一系列工具为程序员使用，能够极大地简化自定义程序的开发，并保持和 ENVI 一致的外观。本文中提供了一系列交互式的例子和实例程序帮助用户理解并开发自定义程序。

2、波段和波谱运算函数

对于用户来说，扩展 ENVI 的功能最简单方法就是使用波段和波谱运算函数。用户可以交互式通过波段和波谱运算表达式对话框输入大多数的波段和波谱运算函数。波段运算函数允许用户从任何波段或是文件中输入数据，处理数据，并输出整个波段。波谱运算函数允许用户从图表或文件中输入波谱数据，处理数据，并输出一个波谱。所有的输入和输出数据选取，数据的获取，波段或波谱函数的调用，结果的输出都由 ENVI 中的波段和波谱函数控制。因此在使用波段或波谱函数进行处理时，用户不需要修改菜单、不必创建参数输入部件，不用执行 I/O 操作，只需在用户编写的函数中提供数据的计算处理功能。

3、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。ENVI 提供了一个采用 IDL 编写的函数和编程工具库，这个库可以进行输入、输出、画图、报告、和文件管理等操作。同时，许多 ENVI 内部的处理函数（如分类程序）也能够在用户函数或批处理模式下供用户使用。ENVI 提供了一系列的复合部件来简化用户部件接口的编写，并使用户的程序和 ENVI 保持相同的外观。

用户函数包括了部件的定义，事件的处理，以及数据处理程序。每一个用户函数都和 ENVI 菜单的相应的按钮相关联，并像其它的 ENVI 函数一样执行。当用户选择菜单中的该选项时，参数部件被创建，并提示用户输入所有处理所需的参数。和部件相联系的事件处理程序管理部件输入。ENVI 提供了小部件事件的自动管理函数，用户不必编写事件处理程序。该函数获得用户输入参数，并传递给用户函数，接着用户函数获取输入影像数据，处理这些

数据，并输出为一个新的影像，绘图、报告或者其它结果。

4、交互式用户程序

交互式程序是 ENVI 交互式应用中自动被调用程序。除了使用 ENVI 中设定的默认方法，用户还可以添加格外的程序。用户能够自定义绘图程序，波谱分析程序，以及移动程序等。

5、自定义文件格式输入

用户可以编写自定义文件输入函数来打开和读取非标准的数据格式。当打开一个不支持的文件格式时，输入程序自动解析头文件信息，并将这些信息加入到可用波段列表中。使用 ENVI 不支持的格式存储的文件中的数据能够通过自定义的读取函数获取。当处理这些数据时，文件能够使用用户自定义的读取函数来获取数据而不必转换成 ENVI 的格式。

6、批处理

批处理模式下，ENVI 提供的功能函数，可以通过 ENVI_DOIT 函数来实现。ENVI_DOIT 函数提供类似于用户函数的处理部分，但不需用户交互。批处理可以通过菜单事件或是 ENVI 非交互模式来启动。

7、ENVI 菜单文件

ENVI 的主菜单文件 envi.men，显示窗口菜单文件 display.men，能够在 ENVI 安装目录下的 menu 文件夹中找到。envi.men 文件中定义了 ENVI 的主菜单项，而 display.men 文件定义了影像显示窗口中的菜单项。两个菜单文件都为 ASCII 文件，其中标明了菜单按钮，下拉菜单定义，以及分隔符。菜单项被选择时要执行的程序也定义在菜单文件中。用户可以依据自己的需求和喜好重新安排菜单或是添加新的菜单项。ENVI 对用户自定义和 ENVI 提供的事件处理程序一样对待，集成用户的事件处理程序非常容易。

8、编译

自定义程序（波段和波谱运算函数，用户函数，交互式用户程序，以及自定义的文件读取工具）必须手动编译或是将这些程序放在 ENVI 安装目录下的 save_add 目录中，该文件夹中的程序在 ENVI 启动时会被自动编译。另一种方法是，如果文件和程序名称吻合，当这些文件在 IDL 搜索路径中时，IDL 能够自动寻找并编译它们。在程序开发和调试时建议在 ENVI 主菜单中选择 Fiel-Compile IDL Module 手动的编译这些文件。一旦这些程序编写完成并工作正常，就可以将它们放到 save_add 目录下，这样每次 ENVI 启动时就能够自动编译。用户可以通过 ENVI 性能选项中改变 save_add 文件夹的位置。批处理程序可以通过 IDL 命令行的.run 命令进行编译。

注意：ENVI RT 不能够编译程序，然而能够通过将编译后的(.sav)文件放在 save_add 目录下，也能够在启动时自动恢复，同样能够执行相应的功能。

9、Toggle Catch

开发用户函数时禁止 ENVI 的捕获错误机制非常有用。当关闭了 ENVI 错误捕获功能后，当程序代码出现错误时，程序将中断执行并提示错误信息。错误信息将打印到 IDL 日志窗口中，变量可以通过使用 ENVI 命令行进行检查。

二、ENVI 编程的介绍

ENVI 提供了大量处理函数供程序员使用，这些函数封装了交互式 ENVI 程序所提供的绝大多数的功能。每个处理程序都是 IDL 的程序或是函数，并可以被任何其它的 IDL 程序所调用。

1、非交互情况下复杂程序的控制

ENVI 处理程序需要大量的用户交互。当用户自己编写代码去调用 ENVI 处理函数时，程序员必需要提供这个处理程序所需的各个参数。因此，大多数 ENVI 程序比典型的 IDL 程序需要更多的关键字。事实上，由于经常需要将如此多的信息传递给 ENVI 处理程序，这些程序通常仅使用关键字而不是与位置有关的参数，这样用户不必以特定的顺序传递信息。

举例来说，假设要在代码中执行一个简单的最大似然分类。该分类操作可以在批处理模式下通过 ENVI_DOIT 加上 CLASS_DOIT 关键字来实现。而当同样的分类在交互式 ENVI 下进行时，用户必须要确定要分类文件名、输入文件的空间和波谱子区、用于训练的 ROIs、这些 ROI 是否从一个输入文件或是从其它文件中获取、一个可能的阈值、是否产生规则影像以及结果是存放在内存或是文件中。在用户编写的程序中调用这个函数时，所有这些参数必须要提供给 ENVI_DOIT。

2、ENVI 和 IDL 环境下文件 I/O 的区别

ENVI 程序中的文件 I/O 和通常的 IDL 程序中的处理差别是很大的。在通常的 IDL 程序中，文件 I/O 需要获得该文件的逻辑单元号（LUN），并使用一些读写程序如，OPENR、READU、OPENW 以及 WRITEU 等对文件进行读写。而在 ENVI 程序中所有文件 I/O 都是由 ENVI 进行内部控制的，这样 ENVI 程序员就没有必要获取 LUN。相反，所有需要输入文件的 ENVI 程序必需要确定一个唯一的文件 ID，FID。FID 本质上是指向数据文件的指针，但它不等同于 LUN。当获取一个文件时，ENVI 通过内部机制获得该文件的 LUN，读写需求的数据，然后释放 LUN。这样 ENVI 无需占用或保存任何 LUN。这种文件 I/O 方式使得 ENVI 中可以同时打开任意数目的文件，尽管 IDL 只提供了 128 个 LUN。

与 IDL 中使用 OPENR 打开文件不同，ENVI 提供了几种不同类型的库函数来打开文件。当文件被打开后，这些函数都会返回该文件的 FID。这个 FID 接着被传递给需要访问数据的 ENVI 函数。同样，当 ENVI 函数的处理结果包括一幅输出影像，不论是存在硬盘或是存在内存中，ENVI 程序返回的都是与该结果相对应的 FID。

3、ENVI 和 IDL 函数库目录

ENVI 和 IDL 都含有一个称为 lib 的目录。这两个目录的功能是不同的，因此选择一个正确的目录来存放用户的程序文件非常重要。

ENVI 的 lib 文件夹中含有用于某些 ENVI 程序的 IDL 代码。这些程序文件只是提供给 ENVI 编程者的一些简单例子，实际上 ENVI 并没有使用这些文件。也就是说，如果用户编辑了该文件夹中的代码，并通过 ENVI 的菜单运行相应的程序，程序中并不会反映这些改动。ENVI 的 lib 目录不在 IDL 的默认搜索路径中，用户可以在 ENVI 的主目录下找到 lib 目录。通常在典型的 Windows PC 上该目录位于：

C:\rsi\idlxx\products\envixx\lib

而 IDL 的 lib 目录中包含了 IDL 所用到的程序文件。例如常用的 IDL 函数 CONGRID（用于改变数组大小）是 IDL 内置的程序，它是由 IDL 语言编写的并存放在 IDL 库目录下。因此，IDL lib 目录总是在 IDL 的默认搜索路径中。IDL 库目录在 IDL 的安装目录下。在典型的 Windows PC 上 IDL 库位于：

C:\rsi\idlxx\lib

当然，用户也可以编辑 IDL 路径使其增加一个新的目录，但 IDL 总是能够找到任何保存在库目录中的文件。

三、ENVI 处理程序的通用关键字

当您熟练使用 ENVI 处理程序时，将会注意到有几个关键基本上所有的程序都是具有的。这些关键字控制了处理所需的基本的文件输入和输出。

1、FID

文件 ID（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，可以被数个 ENVI 程序所使用，来打开或选择文件。ENVI 程序对该文件进行的所有操作都是通过 FID 完成的。但是，如果用户同时使用 IDL 直接读取文件，请注意 FID 和 LUN 不是等同的。

2、R_FID 和 M_FID

ENVI 处理程序所产生的影像结果也包括一个 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 关键字是访问该数据的唯一方法。在掩模处理程序还包括一个 M_FID，或者称为掩模 FID 关键字，用于确定用作掩模波段的文件。

如果打开文件发生错误，FID 将被设置为-1。值为-1 的 FID 是无效的，不能进行进一步的处理。因此在 ENVI 中进行编程时，总是要检查 FID、R_FID 以及 M_FID 是否有效。当遇到一个无效的 FID，通常从当前的处理程序中返回。

3、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，用户必须同时使用 DIMS 关键字确定该文件的空间子集。

DIMS[0]：用于存储指向一个打开的 ROI 区域的指针，仅在 ROI 被定义的时候使用，其它时候设为-1

DIMS[1]：采样的起始位置（一个基于 0 的 IDL 数组下标）

DIMS[2]：采样的终止位置

DIMS[3]：行的起始位置

DIMS[4]：行的结束位置

4、POS

POS 关键字定义了用于处理的波段位置。POS 关键字是一个变长的长整型数组。因为 ENVI 处理的文件可能具有多个波段，而使用 POS 矢量来确定用于处理的波谱子集。波段从 0 开始（band1=0,band2=1,…）；例如，要处理多波段文件的第三波段和第四波段数据，POS=[2,3]。

四、常用 ENVI 函数功能介绍

1、文件管理

ENVI 文件处理函数为程序员提供了相当大的灵活性。有以下的函数可供编程使用，用户可以根据所需的情况选择所需的函数。

ENVI_PICKFILE

ENVI_PICKFILE 函数产生一个提示用户选择文件的对话框。该函数产生的界面和使用 ENVI 主菜单选择 File->Open Image File 一样的界面。该函数并不真正的打开文件，它只是以字符串的形式返回用户所选择的全路径文件名。

ENVI_SELECT

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。该函数产生 ENVI 标准的文件选择对话框，其中包括空间和波谱子区裁剪按钮，以及掩模波段选取按钮。该函数也集成了 ENVI_PICKFILE 的功能，在对话框上提供了文件打开按钮，用户可以通过该按钮打开新的 ENVI 文件。ENVI_SELECT 不仅返回用户所选择文件的 FID，还可以返回进一步处理所需的 DIMS 和 POS 关键字值

ENVI_OPEN_FILE

该函数返回一个文件的 FID，它是打开 ENVI 文件的最直接和简单的方法。默认情况下它将文件信息添加到可用波段列表中，可以使用 NO_REALIZE 可以阻止文件信息加入到可用波段列表中。

注：如果可用波段列表已打开，该关键字无效。

ENVI_FILE_MNG

该函数可以打开、关闭或者删除硬盘上的文件。无需用户交互。

ENVI_GET_FILE_IDS

该函数返回所有当前打开的文件的 FID。

2、打开外部文件格式

ENVI 能够读取相当广泛的数据格式，虽然 ENVI_OPEN_FILE 仅能够打开具有 ENVI 头文件的影像文件。ENVI 也提供了一些特定的处理程序能够打开和返回外部格式的文件：

ENVI_OPEN_DATA_FILE

该函数打开 ENVI 所支持的外部文件（通过关键字指定文件类型）并返回 FID，无需用户交互。

3、获取数据

当影像文件非常大时，不适合使用 IDL 的 READU 命令将它全部读入到内存中。因此，ENVI 提供了两个处理函数能够以小的、易管理的数据块方式读取影像数据。这两个函数也提供了数据逻辑组织，一次一个波段或是一次光谱切片。

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。如果所需的空间数据不止一个波段，必需多次调用该程序以获得该相应波段的数据。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。

4、使用感兴趣区（ROI）进行空间选取

很多 ENVI 函数提供了使用 ROI 进行影像的空间选取选项。在 ENVI 的函数中，DIMS 关键字用于定义空间选取。DIMS 变量的第一个元素称为 ROI 指针，如果它被定义，则表明影像是基于 ROI 进行空间选取的。如果 ROI 指针设置为-1，表示没有使用 ROI。用户可用使用 ENVI_GET_ROI_DIMS_PTR 正确的设置 ROI 指针。

5、生成 ENVI 格式的文件

ENVI 影像格式可能是最简单的数据格式。它是二进制文件，栅格影像数据以二进制数据流方式按 BSQ、BIL 或是 BIP 的存储顺序存储。文件中只有影像数据，头信息没有嵌入到文件中。当使用 WRITEU 程序将二维或三维影像数据写入到磁盘时，IDL 自动生成二进制格式文件。ENVI 格式的影像文件可用使用任意的名称，并且无需扩展名。

同时每一个二进制文件都伴随有一个 ASCII 格式的头文件，这个头文件描述了影像的

基本特征以及附加信息。为了 ENVI 能够识别这个头文件，它必需和影像文件具有同样的文件名，并以.hdr 做为扩展名。

（1） 将影像数据保存到内存

当用户函数的结果是包含在 IDL 数组中的影像数据时，这些数据可用以内存方式被 ENVI 所使用。

ENVI_ENTER_DATA

该函数将IDL数组中的数据输入到可用波段列表中，该程序自动的设置ENVI的头文件，该文件同样也存储在内存中，并返回内存影像的 FID。一旦影像出现的可用波段列表中，它就可以像其它 ENVI 影像一样使用，也能够被存入磁盘。

（2）将影像数据存入硬盘

由于 IDL 的 WRITEU 函数能够产生 ENVI 格式的文件，ENVI 没有提供单独的程序来将 IDL 的数组写入到磁盘。可以直接使用 IDL 的 WRITEU 函数

```
OpenW, unit, 'new_envi_image_file.img', /Get_LUN
WriteU, unit, image_array
Free_LUN, unit
```

注：必需考虑到，要随影像文件同时写入ENVI的头文件。

ENVI_SETUP_HEAD

使用该函数写某个已存入磁盘的影像数据的 ENVI 头文件。使用 OPEN 关键字，允许将影像文件输入到可用波段列表。如果没有给 ENVI_SETUP_HEAD 程序设置 OPEN 或是 WRITE 关键字，那么 ENVI 头文件只是在内存中创建（可以使用 ENVI_FILE_QUERY 获取文件信息）。ENVI_SETUP_HEAD 函数也能够返回磁盘上影像文件的 FID。

（3）从已存在的 ENVI 文件中创建新文件

CF_DOIT

这是第三个用于创建 ENVI 格式文件的函数，尽管它只能用于 ENVI 中已经打开的文件。使用该函数通过已有的 ENVI 文件创建一个新的 ENVI 格式的文件。集成到新文件中去的影像可以是 ENVI 中已经打开的磁盘文件或是内存文件，结果可以保存为文件也可以存放在内存中。

第二章 波段和波谱运算函数

波段函数和波谱函数为用户提供了一个简单的编程接口，能够为 ENVI 添加新的处理功能。运算函数不需要修改菜单，不需创建参数设定界面，不用执行 I/O，或是其它的处理过程所必须的操作。ENVI 为用户做了这一切，用户只需构建处理函数即可。

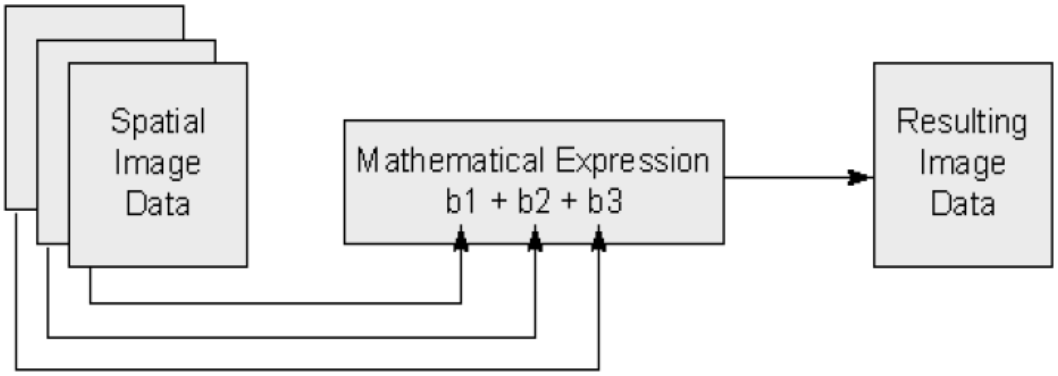
一、波段运算

1、波段函数基础

ENVI 波段运算工具能够调用用户编写的程序进行定制的处理。波段运算工具用来定义输入的波段或文件，并调用用户编写的函数，最后将结果输出到文件或是内存中。

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该波段运算函数。通过波段运算对话框为变量赋值。

下图通过三个波段的相加说明了波段运算的处理过程。表达式中的每一个波段都对应于一个输入的影像波段，对这三个波段求和并输出结果影像。这个表达式中的每个变量不仅可以对应于单一波段，也可以对应与一个文件。例如：在表达式 b1+b2+b3 中，如果 b1 映射为文件而 b2,b3 映射为单一波段则结果为 b1 所对应的文件的所有波段分别和 b2、b3 进行求和。



例如：要执行一个名为 BM_RATIO 的函数，它具有两个输入波段，在 Band 函数表达式中输入以下表达式：

Bm_ratio(b1,b2)

该函数定义如下：

Function bm_ratio,b1,b2

在波段运算中函数中执行的操作和波段运行表达式执行的操作具有同样的局限性。输入数据时分块的，因此类似 min()、max()等函数是无效的，因为它们只返回当前数据块的最小值和最大值，而不是整个输入波段的最小值和最大值。

2、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
```

注：为了兼容分块处理，自定义的波段运算函数应该避免需要整个波段同时在内存中的情况。

3、编译波段运算函数

一旦函数完成后，结果.pro 或.sav 文件需要放到 ENVI 安装路径下的 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 ENVI 主菜单中选择 File—Compile IDL Module 进行手动编译。

注意：ENVI RT 不能编译.pro 文件，只能使用.sav 文件。

4、波段运算例子

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)

```
;执行(b1+b2)/(b1-b2)的运算并检查是否被 0 除
;函数定义为两个输入波段 b1 和 b2 以及一个 check 关键字
function bm_ratio,b1,b2,check=check
    ;计算差值
    den=float(b1)-b2
    ;如果设置了 check 关键字，检查被 0 除问题
    if(keyword_set(check)) then ptr=where(den eq 0.,count) $
    else count=0
    if (count gt 0) then den [ptr]=1.0
    ;继续计算比率结果
    result=(float(b1)+b2)/den
    if(count gt 0) then result[ptr]=0.0
```

```
    ;返回结果
    return,result
end
```

下列步骤列出了如何执行该函数：

（1）保存文件为 bm_ratio.pro 并放置到 save_add 目录下

（2）启动或重新启动 ENVI

（3）打开一个文件包含两个波段

（4）选择 Basic Tools -> Band Math

（5）执行两个波段的比值，不检查是否被 0 除，在表达式文本框中输入：

```
Bm_ratio(b1,b2)
```

（6）执行两个波段的比值，并检查是否被 0 除，输入下面的式子：

```
Bm_ratio(b1,b2,/check)
```

二、波谱运算

1、波谱运算基础

ENVI 的波谱运算功能可以调用用户编写的程序执行自定义处理。波谱运算接口用于定义波谱或文件用作输入，调用用户函数，并将结果写入到绘图窗口、文件或内存中。用作输入的波谱必需显示在绘图窗口中。波谱运算通过映射变量到波谱或文件来获取数据。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。

下图通过进行三个波谱的相加来说明波谱运算的过程。表达式中的每一个变量都被映射为一个输入波谱，对这三个波谱求和并将结果输出到绘图窗口。表达式中的变量也可以映射为文件而不仅是单一波谱，在这种情况下，结果将是一幅新的影像。例如，在表达式 s1+s2+s3 中，如果 s1 映射为一个文件，而 s2 和 s3 映射为单一波谱，那么结果影像将是 s1 代表的影像加上 s2 和 s3。

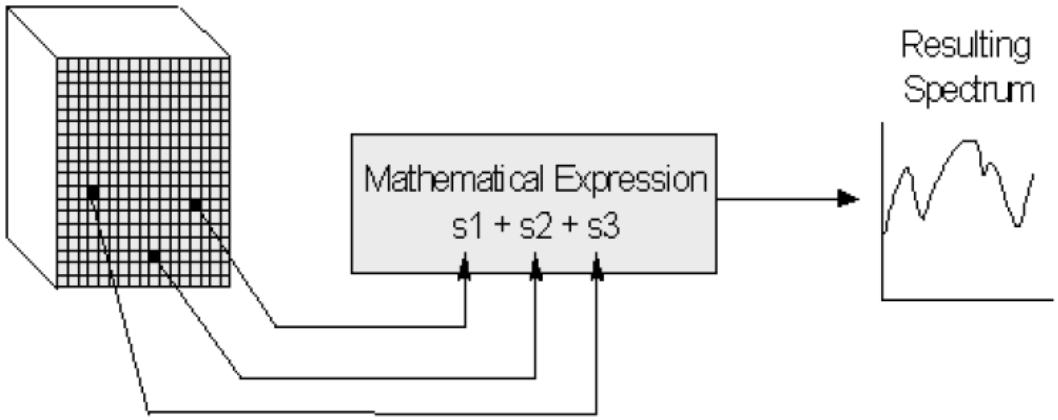


Figure 2-2: Spectral Math Processing — Addition of Three Spectra

例如：要执行一个称为 SM_SCALE 的函数，该函数有两个输入波谱，在波谱表达式文本框中输入：

Sm_scale(s1,s2)

该函数定义为：

Function sm_scale,s1,s2

2、编写波谱运算函数

在波谱运行表达式中执行的波谱函数，编写非常简单。该函数接受输入波谱，处理数据，并返回结果。函数定义如下：

Function sm_func, s1,[s2,….,sn, parameters and keywords]

Processing steps

Return,result

End

波谱运算函数的输出为单一波谱或和输入具有同样波段的波谱。但一个输入参数被映射为文件，所有行的波谱将一次进行处理。

3、编译波谱运行函数

一旦完成函数的编写，推荐用户将程序结果.pro 或.sav 文件放置到 ENVI 安装路径下的 save_add 目录下，这样 ENVI 在启动时会自动编译或恢复这些函数。同时，该函数也可以通过在 ENVI 的主菜单上选择 File-> Compile IDL Module 进行手动的编译。

4、波谱函数的例子

要执行如下的运算：s1 / s2

;执行如下计算

;S1/S2 并检查是否被 0 除

```
;函数定义为两个输入光谱数据，S1 和 S2，以及一个关键字 check
function sm_ration,s1,s2,check=check
    ;如果定义了 check 关键字就检查被 0 除的情况
    ;首先找到 S2 中为 0 的数据位置
    if(keyword_set(check)) then ptr=where(s2 eq 0.,count) $
    else count=0
    ;将找到的数据位置赋值为 1.0
    if(count gt 0) then s2[ptr]=1.0
    ;执行运算
    result=float(s1)/s2
    ;将 0 值位置重新赋值为 0
    if(count gt 0) then result[ptr]=0.0
    ;返回处理结果
    return,result
end
```

执行下列步骤，来运行本例：

（1）将函数保存为 mfspec.pro 并放置到 save_add 目录下

（2）启动或重起 ENVI

（3）打开一个多波段的文件

（4）显示影像，并通过在影像窗口菜单下选择 Tools-> Profiles->Z Profile 绘制两个波谱

（5）在波谱工具下拉菜单中选择波谱运算

（6）执行两个波谱的比率，而不检查是否被零除，在表达式对话框中输入下面的内容：

Sm_ratio(s1,s2)

（7）执行两个波谱的比率，检查是否被零除，在表达式对话框中输入下面的内容：

Sm_ratio(s1,s2,/check)

第三章 批处理模式

许多 ENVI 影像处理程序无需用户交互，它们能够在批处理和交互式环境下运行。执行批处理在于构建一个程序包含所有的程序调用和合适的参数。批处理程序能够从菜单系统中以最小 GUI 输入的方式启动，也可通过没有 GUI 输入的方式脱离菜单系统。第一种方式允许将各种处理程序联系在一起并由简单的菜单选择启动。而第二种方式可以对大量的文件进行处理。

一、ENVI 的批处理模式

以批处理模式运行 ENVI 能够让用户在命令模式下使用 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 工作但偶尔需要用到 ENVI 的函数；

（2）用户希望创建定制的应用程序其中混合了 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 ENVI 库函数，有必要了解一下 ENVI 程序的结构。

ENVI 功能函数分散在大约 50 个小的 IDL save 文件中，这些二进制的文件包括数据和编译后的程序。这些 save 文件存放在 ENVI 安装路径下的 Save 目录下。ENVI 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。在典型的 Windows PC 上这些文件存放在：C:\rsi\idlxx\products\envixx\save 目录下。在典型的 UNIX 系统中，这些文件位于/usr/local/rsi/idl_x.x/products/envi_x.x/save。

在批处理模式下运行 ENVI 需要先恢复核心的 save 文件，而后，一个称为 ENVI_BATCH_INIT 的特定 ENVI 函数被调用，该函数开启批处理模式。该步处理称为初始化批处理。

二、混合批处理模式

ENVI 始终是一个 IDL 程序，如果用户使用运行 ENVI 的 IDL 时段，用户将能够访问所有 ENVI 程序和函数。这种状态通常称为混合批处理模式，因为用户能够使用 ENVI 特有的库函数而无需初始化批处理模式。这种状况既带来了便利，也带来了问题。举例来说，如果用户在 IDL 命令行下运行的程序产生了新的影像波段，能够通过 ENVI_ENTER_DATA 直接将这些数据输入到可用波段列表中。但如果 IDL 程序崩溃，那么当前的 ENVI 环境将会整个崩溃。在编写 ENVI 用户函数的时候混合批处理模式非常有用，因为它模拟了代码执行的最终环境。但是在真正运行批处理程序时，推荐用户开启一个单独的 IDL 时段，并进行批处理的初始化。

三、批处理模式初始化

初始化 ENVI 批处理模式需要恢复 ENVI 的二进制 save 文件，并调用 ENVI 的命令 ENVI_BATCH_INIT。

1、启动 ENVI；

2、从 ENVI 的主菜单条上选择 File ->Preferences；

3、点击 Miscellaneous 选单，并确定 Exit IDL on Exit from ENVI 选项设置为 No；

4、退出 ENVI，如果需要退出 IDL。

警告：不要在已经运行交互式 ENVI 的 IDL 时段中启动 ENVI 的批处理模式。重新启动一个 IDL 时段并以批处理模式初始化 ENVI。

5、启动一个新的 IDL 时段，并输入以下命令：

```
IDL>ENVI, /restore_base_save_files
```

如果忘了设置关键字 RESTORE_BASE_SAVE_FILES，最终会启动正常的 ENVI 程序；

6、接着，调用 ENVI_BATCH_INIT 程序

调用 ENVI_BATCH_INIT 等于启动了一个没有 GUI 的新 ENVI 时段。如果已经是批处理状态，后面再调用 ENVI_BATCH_INIT 命令将会被忽略。

下面的是批处理初始化的例子：

```
pro bt_init
```

```
  envi,/restore_base_save_files
  envi_batch_init,log_file='batch.log'
   ;Batch processing would go here
  envi_batch_exit
end
```

四、离开批处理模式

如果在结束批处理后，你还将继续在 IDL 时段中工作，正确的退出批处理模式是非常重要的。因为像 ENVI 这样复杂的软件，运行时会创建许多不同的变量、一些公共模块、结构、指针。当退出 ENVI 时，必须要正确的删除这些变量并释放内存。这在用户要继续在同一 IDL 时段中工作尤为重要。ENVI_BATCH_EXIT 退出批处理的方式和通过 ENVI 主菜单上选择 File-> Exit 退出 ENVI 的效果一样。同样，使用该命令退出 ENVI 后，ENVI 时段使用的 License 也被释放。

五、编写批处理程序

ENVI 批处理的主要目的是允许用户进行无需交互的 ENVI 处理。许多用户也会发现也能够通过使用 ENVI 库函数很方便的在自己的 IDL 程序中加入新的功能。

如果 ENVI 中已经有了您想要实现的功能，为什么不使用它呢？当然为了获取这些库函数，运行程序的 IDL 时段必须在批处理模式下。推荐用户在 IDL 程序中初始化批处理模式，并在处理完成后使用 ENVI_BATCH_EXIT 函数退出批处理模式，以释放 ENVI 所使用的资源和内存。

例子：编写一个简单的 ENVI 批处理程序查看 DEM：VIEW_DEM

本例中，首先提示用户选择一个 DEM 文件，读入数据后，同时显示 DEM 以及它相关的阴影影像。不用自己编写阴影生成算法，我们只需使用 TOPO_DOIT 关键字调用 ENVI 地形分析功能来计算获得阴影影像。

```
pro view_dem
  ;严格编译器要求
  compile_opt idl2
  envi,/restore_base_save_files
  envi_batch_init
  ;选取 DEM 文件
  dem_file=dialog_pickfile(title='select a DEM')
  if(dem_file eq '') then return
  ;打开 DEM 文件
```

```
  envi_open_file,dem_file,r_fid=dem_fid
  ;获取文件信息
  envi_file_query,dem_fid,ns=ns,nl=nl
  proj=envi_get_projection(fid=dem_fid,pixel_size=pixel_size)
  dims=[-1L,0,ns-1,0,nl-1]
    ;生成阴影影像
  envi_doit,'TOPO_DOIT',azimuth=15.0,bptr=[2],dims=dims,$
      elevation=45.0,fid=dem_fid,in_memory=1,pos=[0],$
      r_fid=shaded_fid,pixel_size=pixel_size
  dem=envi_get_data(fid=dem_fid,dims=dims,pos=[0])
  shaded=envi_get_data(fid=shaded_fid,dims=dims,pos=[0])
  ;同时显示 DEM 和阴影影像
  window,/free,xsize=(2*ns),ysize=nl
  tvscl,dem,order=1
  tvscl,shaded,ns,0,order=1
  envi_batch_eixt
end
```

六、在批处理模式中记录 ENVI 的日志信息

可能 ENVI 批处理模式最主要的作用在于无需交互的处理数据文件。例如：如果您需要使用 ENVI 分别处理数百个影像文件，您可以编写一个 IDL 程序来找到所有的文件，打开它们，执行所有的 ENVI 处理，并将结果保存到磁盘上。所有的处理都能够在无人值守下进行，用户所做的只需在 IDL 程序中调用 ENVI 的批处理模式。

当在无人值守的情况下使用 ENVI 批处理模式进行处理时，要确保以下两点：（1）当产生错误或出现错误信息时，批处理不会被中断；（2）系统中重要的信息被收集存储并在处理完成后能够察看。

在初始化批处理的时候，用户可以使用 LOG_FILE 关键字来定义一个日志文件。这个关键字传递给 ENVI 一个文件名，该文件用来写入各种错误和提示消息。

```
IDL>ENVI, /RESTORE_BASE_SAVE_FILES
IDL>ENVI_BATCH_INIT, LOG_FILE='test_batch_log.txt'
```

在运行 ENVI 批处理程序时，批处理日志收集任何系统产生的消息。用户也可以使用 ENVI_ERROR 函数将自己的信息写入日志文件。

七、批处理的有用提示

可能每个用户对批处理都有不同的需求，但下面的建议是非常有用的：

1、在 IDL 中运行批处理代码，而不要在混合模式下运行。在执行批处理程序的 IDL 时段中运行正常的 ENVI 将会导致一些库函数挂起并等待用户输入。

2、使用 IDL 的函数 FILE_SEARCH 列出要处理的文件列表，推荐使用全路径名称。

3、使用 IDL 的字符串操作如 STRMID、STRPOS 以及 STRSPLIT 来根据输入文件构建输出文件名。

4、由于 IDL 的编译器不能识别 ENVI 库函数，对它们的任何引用都会被错误的解释为一个变量名。如果您的 IDL 的代码中使用[]来进行数组元素的提领，那么您可以使用 COMPILE_OPT STRICTARR 语句来解决这个问题。如果您使用（）来提领数组元素，您必需使用 FORWARD_FUNCTION 来声明 ENVI 的函数，这样 IDL 编译器才不会把它们当作变量。

八、为批处理创建一个快捷方式

如果用户经常使用批处理模式，将批处理模式初始化的命令放置到一个 IDL 文件中，这样就可以使用一个简单的命令来启动批处理模式。这儿有个简单的例子：

```
PRO ENVI_BATCH
    ENVI, /RESTORE_BASE_SAVE_FILES
    ENVI_BATCH_INIT
END
```

现在用户可以在 IDL 的命令行中输入 ENVI_BATCH 来快速启动批处理模式。

八、批处理程序的例子

1、文件信息的统计

本例使用非交互的批处理来计算特定文件的基本统计信息。首先 ENVI 核心文件被恢复，并启动 ENVI 批处理模式。接着通过使用 ENVI_OPEN_FILE 打开文件并返回 FID。该 FID 被传递给统计处理程序，ENVI_STATS_DOIT。当处理完成后，使用 ENVI_BATCH_EXIT 退出 ENVI 时段。

本例代码如下：

```
;说明在 ENVI 批处理模式下中如何对文件进行统计
pro bstats1
;首先要恢复 ENVI 的核心文件并开启批处理模式
```

```
envi,/restore_base_save_files
envi_batch_init,log_file='batch.log'
;选择文件
file=dialog_pickfile(title='请选择一个文件',/read)
;打开输入文件,获得 FID
envi_open_file,file,r_fid=fid
;如果打开文件失败，退出程序
if (fid eq -1) then begin
    envi_batch_exit
    return
endif
;获得文件的信息
envi_file_query,fid,ns=ns,nl=nl,nb=nb
dims=[-1,0,ns-1,0,nl-1]
pos=lindgen(nb)
;计算统计
envi_doit,'envi_stats_doit',$
    fid=fid,pos=pos,dims=dims,$
    DMIN=dmin,DMAX=dmax,MEAN=mean,$
    STDV=stdv,COMP_FLAG=1
print,dmin,dmax,mean,stdv
;退出 ENVI
envi_batch_exit
end
```

一些其它的统计信息也可以通过 ENVI_STATS_DOIT 实现。

通过下面的步骤来执行该例：

（1）根据当前机器的情况修改路径和文件名。

（2）启动 IDL（不用启动 ENVI）

（3）在 IDL 命令行中输入以下命令：

```
.compile c:\my_path\bstats1
```

(4) 在 IDL 命令行中输入：btstats1 来执行该程序。

2、饱和度拉伸（非交互）

下面的例子对一幅 RGB 影像执行饱和度拉伸。本例实现的方法是在无交互的情况下执行。这个批处理程序使用了一些 ENVI 的处理函数。进行的处理和相应的函数罗列如下：

（1）打开 RGB 影像文件，ENVI_OPEN_FILE

（2）对 RGB 影像进行 2%线性拉伸，STRETCH_DOIT

（3）将拉伸过的 RGB 影像变换到 HSV 空间，RGB_TRANS_DOIT

（4）对饱和度波段进行高斯拉伸, STRETCH_DOIT

（5）将 HV 以及拉伸后的饱和度波段转换回 RGB 空间, RGB_ITRANS_DOIT

ENVI 使用 ENVI_BATCH_INIT 函数初始化批处理模式。STRETCH_DOIT、RGB_TRANS_DOIT 和 RGB_ITRANS_DOIT 函数，在被 ENVI_DOIT 调用时会被自动恢复。

示例代码如下：

```
;对图像进行饱和度拉伸，不需用户干预
;对 RGB 图像进行 2%线性拉伸
;将 RGB 图像变换的 HLS 空间
;对饱和度进行高斯拉伸
;将处理结果重新变换回 RGB 空间
pro satstrch
    ;恢复 ENVI 的核心文件
    envi,/restore_base_save_files
    ;开启批处理模式
    envi_batch_init,log_file='batch.log'

  in_name=dialog_pickfile(title='请选择一个文件',/read)
  out_name='c:\new_rgb'
  tmp1_name='c:\tmp1'
  tmp2_name='c:\tmp2'
  tmp3_name='c:\tmp3'
  envi_open_file,in_name,r_fid=fid
  if(fid eq -1) then begin
      envi_batch_exit
      return
  endif
   envi_file_query,fid,ns=ns,nl=nl,bnames=bnames
   dims=[-1,0,ns-1,0,nl-1]
   pos=[0,1,2]
   envi_doit,'stretch_doit',$
       fid=fid,pos=pos,dims=dims,$
       out_name=tmp1_name,method=1,out_dt=1,$
       i_min=2.0,i_max=98.0,range_by=0,$
```

```
        out_min=0,out_max=255,in_memory=0,$
        r_fid=st_fid
  if(n_elements(st_fid) eq 0) then begin
      envi_batch_exit
      return
  endif
;将拉伸后的数据转换到 HLS 空间
  envi_doit,'rgb_trans_doit',fid=[st_fid,st_fid,st_fid],$
    pos=pos,out_name=tmp2_name,dims=dims,r_fid=hls_fid,$
    hsv=0,in_memory=0
  if(n_elements(hls_fid) eq 0) then begin
    envi_batch_exit
    return
  endif
  ;对饱和度进行高斯拉伸
  envi_doit,'stretch_doit',$
      fid=hls_fid,pos=[2],dims=dims,$
      method=3,range_by=0,i_min=0.0,$
      i_max=100.0,stdv=2.0,out_dt=4,$
      out_min=0.0,out_max=1.0,in_memory=0,$
      r_fid=gst_fid,out_name=tmp3_name
  if(n_elements(gst_fid) eq 0) then begin
        envi_batch_exit
        return
  endif
  out_bname='satstrch('+bnames[pos]+')'
  envi_doit,'rgb_itrans_doit',$
        fid=[hls_fid,hls_fid,gst_fid],pos=[0,1,0],$
        out_name=out_name,dims=dims,hsv=0,$
        out_bname=out_bname,in_memory=0
;关闭输入文件并删除硬盘上的零时文件
  envi_file_mng,id=fid,/remove
  envi_file_mng,id=st_fid,/remove,/delete
  envi_file_mng,id=hls_fid,/remove,/delete
  envi_file_mng,id=gst_fid,/remove,/delete
```

```
;退出 ENVI
envi_batch_exit
end
```

通过下面的步骤来执行该例：

（1）根据当前机器的情况修改路径和文件名。

（2）启动 IDL（不用启动 ENVI）

（3）在 IDL 命令行中输入以下命令：

```
.compile c:\my_path\ satstrch
```

(4) 在 IDL 命令行中输入：satstrch 来执行该程序。

（5）打开 ENVI，查看处理结果。

第四章　用户函数

一、用户函数介绍

用户函数是简单的 IDL 代码，可以调用 ENVI 处理程序，并能够通过 ENVI 的菜单进行交互式的访问。在设计用户函数时，用户可以选择非交互方式，或是使用 ENVI 的复合组件进行简单的用户界面的设计并保持和 ENVI 一致的界面，或是通过 IDL 部件工具创建新的界面。如果使用 ENVI 的复合组件，可以让 ENVI 自动进行事件管理。

用户函数允许用户为 ENVI 添加新的功能并通过 ENVI 菜单进行访问。可以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该函数对应的菜单项时，将会执行这些函数，就如同 ENVI 的自身函数一样。用户函数和 IDL 程序没有什么区别，可以称之为 ENVI 的程序。也就是说，它们和批处理模式也很接近，除了无需初始化批处理。

用户函数是部件事件的处理程序。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要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中的信息，这个参数也必须要加上。

用户函数程序文件可以是.pro 或是.sav 文件，同其它的 IDL 程序一样，也可以包括调用的 C 或是 Fortran 代码。此外，将用户函数放入到 ENVI 安装目录下的 Save_add 目录中，在 ENVI 启动时用户函数可以被自动的编译或恢复。同时，加入到 ENVI 中的用户函数的代码可以随时进行修改，并在 ENVI 的 session 中重新编译，而无需重新启动 ENVI。

二、修改 ENVI 的菜单

1、ENVI 菜单系统简介

ENVI 的菜单系统，包括主菜单和显示窗口菜单，是由 ENVI 安装目录下 MENU 目录下的 envi.men 和 display.men 这两个 ASCII 码文件定义的。

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 主菜单中的选项，display.men 文件定义了显示窗口菜单中的选项。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。

2、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。在文件的顶部有些介绍的注释文本。注释结束后，就是如下的文件的结构：

```
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，如此类推。

{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中。

{envi_menu_event}第三个大括号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，同时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，用户值是不会使用的，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。

三、编写一个用户函数的实例

通过本例用户将更加熟悉 ENVI 的菜单修改。我们在 ENVI 菜单中创建一个新的菜单项，并定义一个简单的用户函数。这个用户函数可以通过在 ENVI 菜单中选择这个新的菜单项来进行调用。

(1)、修改 ENVI 的菜单，用文本编辑器打开 envi.men 文件，滚动到底部，添加如下部分：

0{MyFunctions}

1{Basic File Info}{not used}{file_info}

保存菜单文件。如果 ENVI 已经打开，关闭并重新启动 ENVI，检查看您定义的新菜单选项是否出现在菜单上。

（2）编写用户函数

在运行 ENVI 的 IDL 开发环境中，新建一个 Editor 创口，输入如下代码：

```
pro FILE_INFO,event

  compile_opt strictarr
  envi_select,title='请选择一个文件',fid=in_fid

  if(in_fid eq -1) then return

  envi_file_query,in_fid,ns=ns,nl=nl,nb=nb,fname=fname

  openr,unit,fname,/get_lun

  info=fstat(unit)

  free_lun,unit

  print,'你选择的是：',fname
  print,'number of samples=',ns
  print,'number of lines=',nl
  print,'number of bands=',nb
  print,'file size in bytes=',info.size
```

end

将该程序保存，并编译。

（3）、执行用户函数

在 ENVI 中，点击新添的菜单选项，运行上面编写的用户函数。在 IDL 日志窗口中，将会输出文件的基本信息。

四、为用户函数添加小部件

用户函数可以采取各种形式，也能用于各种不同的目的。有些用户函数即使是通过 ENVI 菜单交互式的选取执行，它们也无需用户交互。例如，一个用户函数可以用来执行批处理。但是，用户函数最适合的是进行复杂处理并要求用户输入一些参数。虽然在命令行收集参数也是可行的，但是这不适合 ENVI 用户，因为 ENVI 参数过多不适合在命令行输入。使用部件窗口来交互式收集用户输入就相当简单和更加直接，这也是大多是 ENVI 函数所采用的模式。

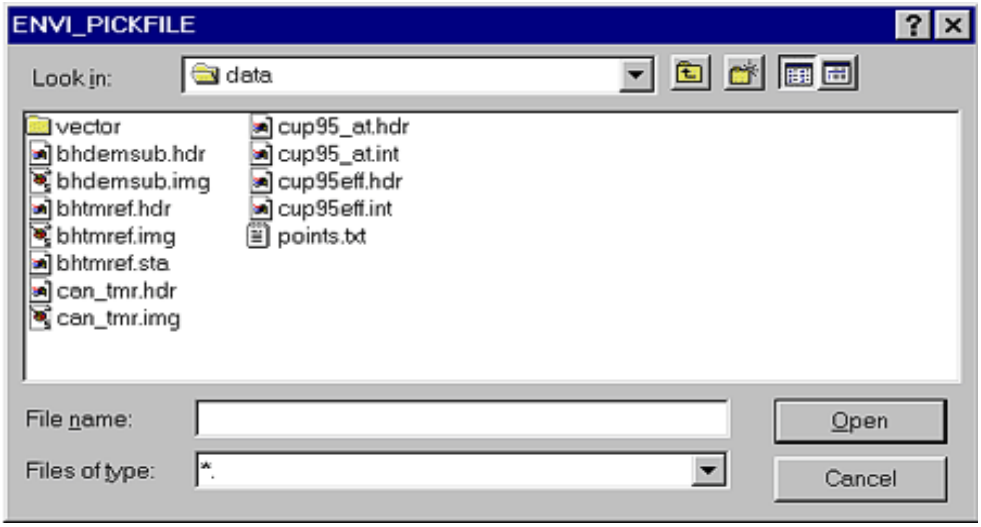
使用小部件收集用户输入比通常的 IDL 程序中使用 Widget 要容易，因为用户无需编写事件处理程序，ENVI 能够自动管理部件事件。此外 ENVI 复合部件包括在 ENVI 的函数中，这样就能够很方便的创建和 ENVI 类似的界面。

五、可用的 ENVI 部件

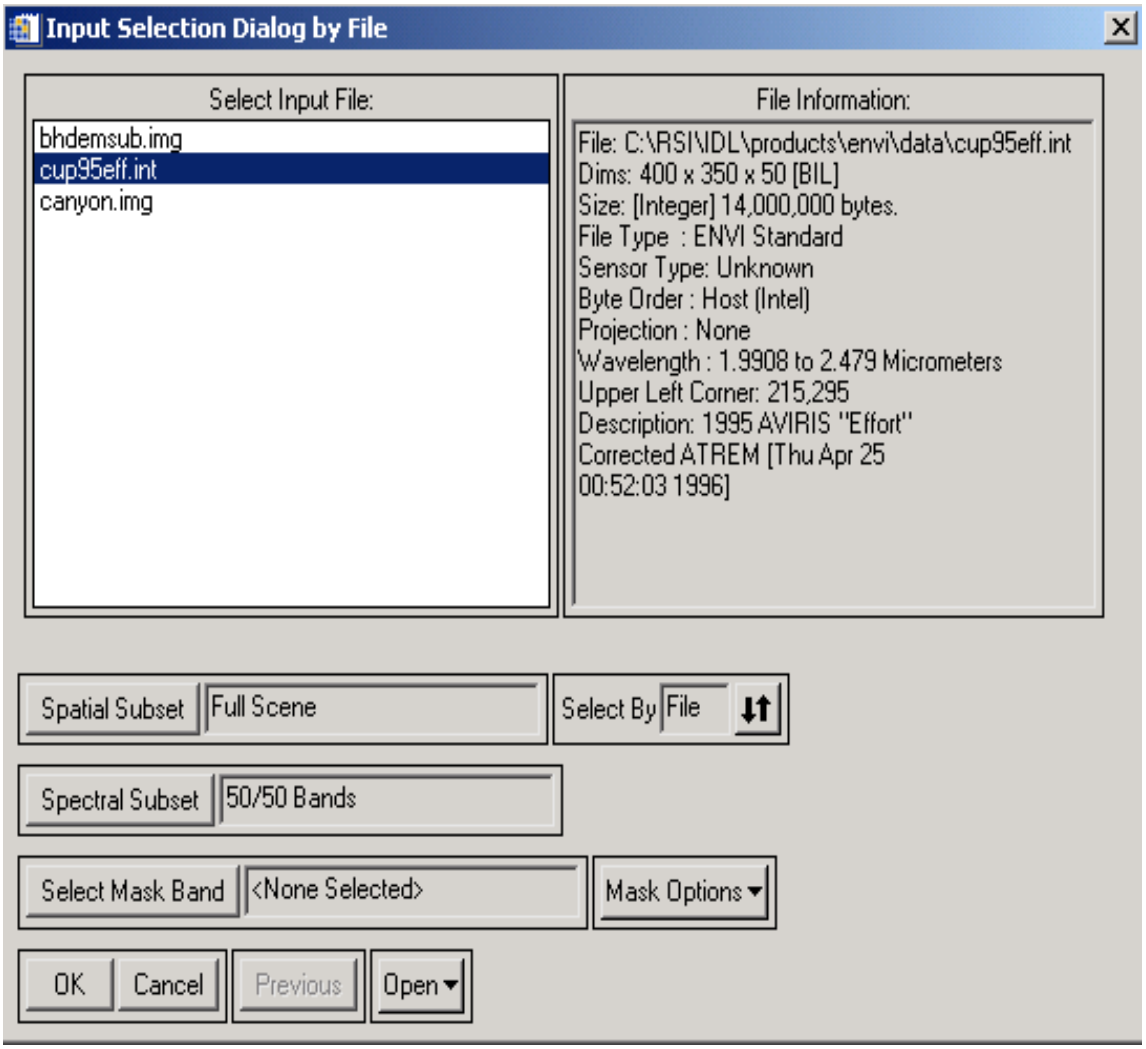
ENVI 提供了 20 多种复合的部件，可以为用户函数所用。大多数的函数以 WIDGET_ 开头。

ENVI_PICKFILE

用于从硬盘上选择一个文件。尽管常用于选取影像文件，它也可以用来收集任意类型的文件名。

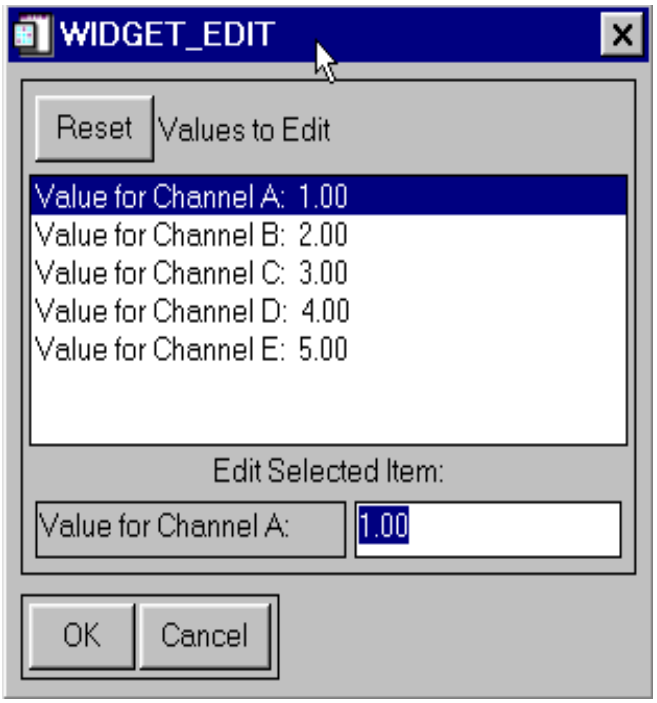


ENVI_SELECT



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。

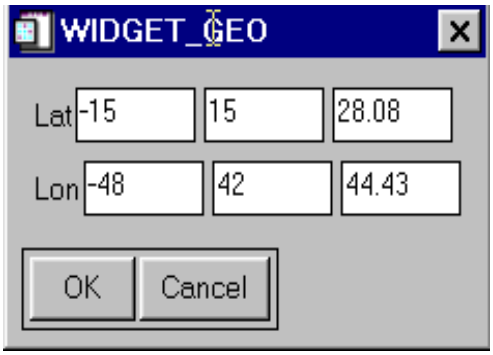
WIDGET_EDIT



提供了一个能从列表中选取项目的部件。

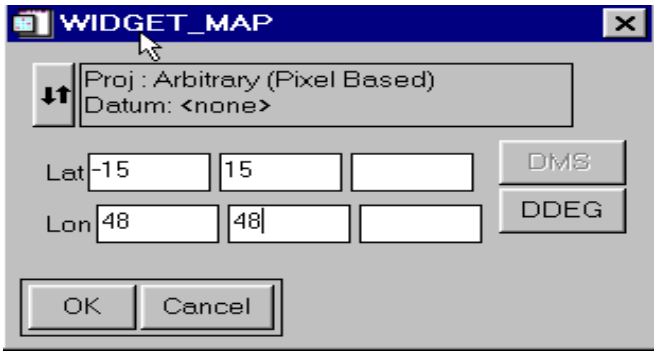
WIDGET_GEO

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。

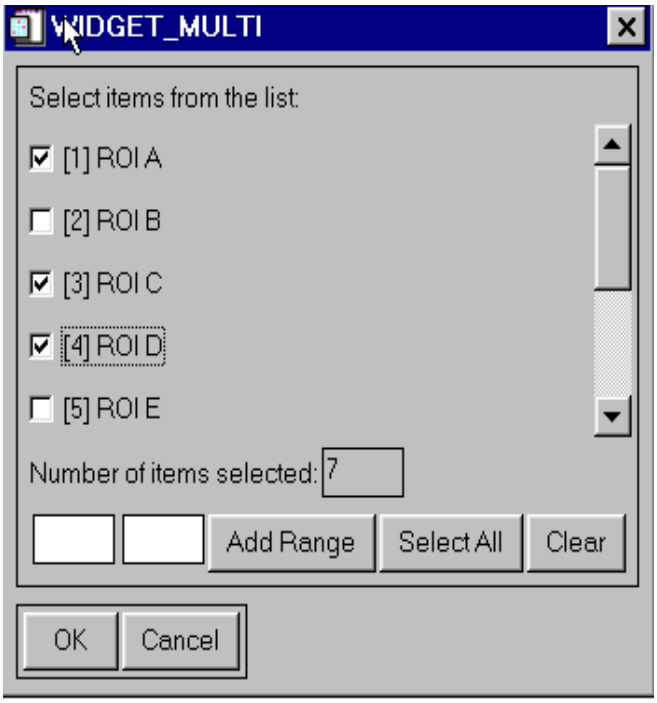


WIDGET_MENU

一个复合部件，用来生成一些复选或非复选的按钮。

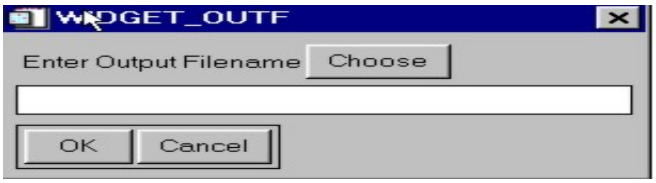
WIDGET_MULTI

用于多项选择的复合部件，在 ENVI 分类的 ROI 选取中常用



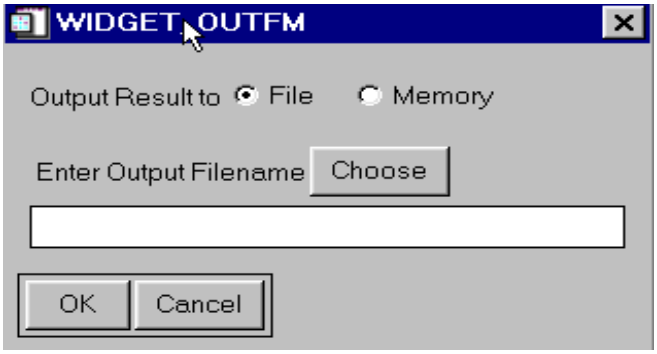
WIDGET_OUTF

用于选择一个输出文件名。



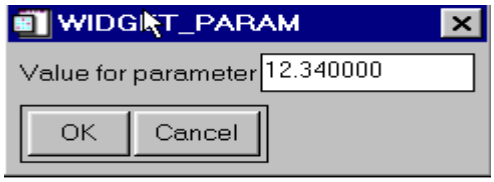
WIDGET_OUTFM

用于选择一个输出文件名或是输入到内存



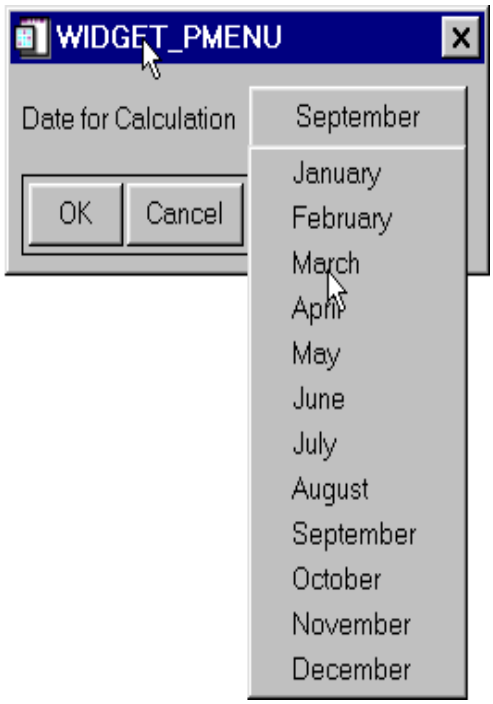
WIDGET_PARAM

用于输入数字参数。



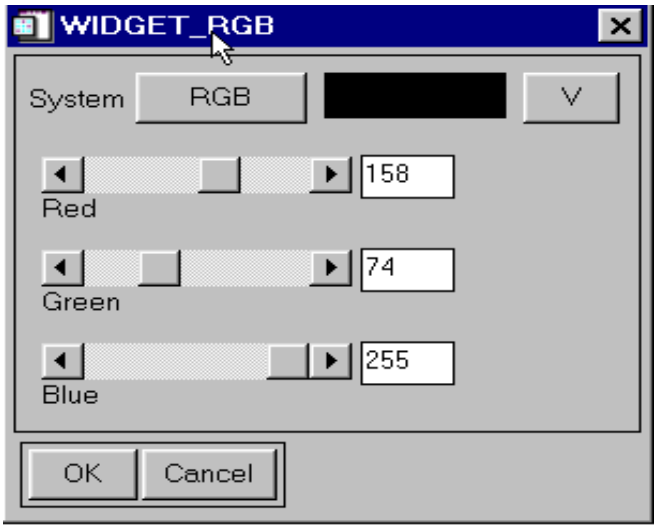
WIDGET_PMENU

提供下拉菜单



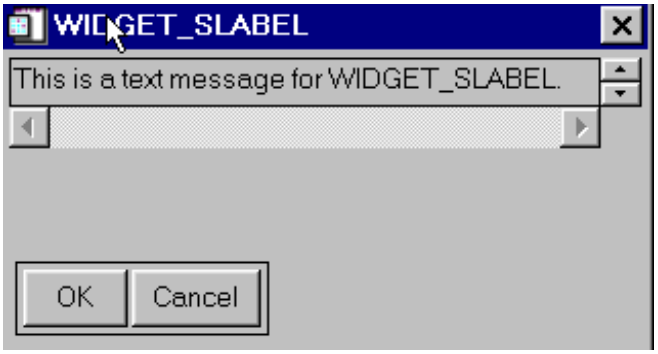
WIDGET_RGB

用于修改 RGB 颜色值的复合部件。



WIDGET_SLABEL

用于显示文本信息，并具有滚动条的复合部件。



WIDGET_SLIST

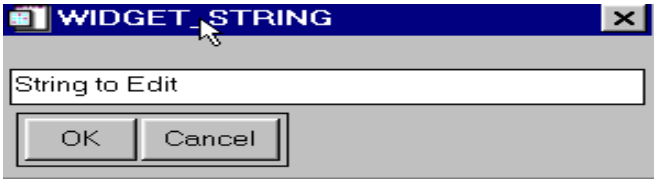
用于创建列表的复合部件，被选择的项能够在文本框中显示。

WIDGET_SSLIDER

一个使用滑块设置数字值的复合部件。

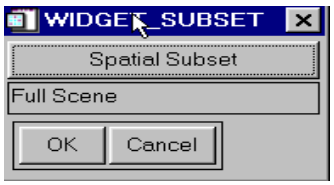
WIDGET_STRING

用于输入字符串的复合部件。



WIDGET_SUBSET

用于确定空间裁剪的 ENVI 标准部件，该部件上有一个空间裁剪按钮，通过该按钮能够启动标准的空间裁剪对话框。



六、小部件事件自动管理程序

在通常的 IDL 程序中，程序员必须要为程序中使用的小部件编写事件处理程序。对刚使用 IDL 的用户来说，编写事件处理程序是一个比较大的难题。因此为了方便在 ENVI 用户函数中使用部件，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。用户可以通过使用下面的两个 ENVI 函数进行事件自动管理。

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，如果要创建自动事件管理的部件构架，必须通过 WIDGET_AUTO_BASE 创建顶级 BASE，GUI 中的其它 BASE 仍然使用原来的 WIDGET_BASE 函数创建。使用 WIGET_AUTO_BASE 函数创建的顶级 BASE 是自动列对齐，居中并且是模态化的。不同于 WIDGET_BASE，WIGET_AUTO_BASE 仅接受几个关键字来控制它的属性。WIDGET_AUTO_BASE 接受的关键字有：GROUP、TITLE、XBIG 以及 YBIG。

AUTO_WID_MNG

在通常的 IDL 部件程序中，一旦 GUI 被定义，XMANAGER 程序被调用进行部件的注册并进行部件事件的检测，这种情况下程序员要对不同的事件进行管理。而在自动事件管理的 ENVI 程序中，ENVI 程序员无须调用 XMANAGER 程序，相反，特定的 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值。AUTO_WID_MNG 函数自动在 GUI 的底部加上了”OK”和”Cancel”两个按钮。

通过如下的方式调用该函数：result=AUTO_WID_MNG(TLB)

表达式中的 TLB 必须是通过 WIDGET_AUTO_BASE 创建的顶 BASE。当用户函数的 GUI 关闭后，RESULT 将被设置为一个含有 GUI 中使用部件值的结构。该结构中的每一个标识名称是通过用户值定义的。该用户值是在 GUI 定义的时候分配给每一个 ENVI 复合部件的。

这种技术让 ENVI 用户函数看起来非常简单。

```
Pro MY_USER_FUNCTION ,Event
    TLB=WIDGET_AUTO_BASE(…)
    Lst_parameter=WIDGET_PARAM(TLB,uvalue=’param1’…)
    Result=AUTO_WID_MNG(TLB)
    Do the processing…
End
```

在这个例子中，result.param1 的值便包含了用户在 WIDGET_PARAM 部件中输入值。每一个 ENVI 部件返回值的信息在该函数的帮助中有具体的介绍。

例子：构建一个简单的自动事件管理程序

本例的目的是使用户熟悉图形用户界面的构建，并使用结果结构。这个用户函数获得两个数字参数，并提示用户选择加还是乘。

我们称这个用户函数为 TEST_WIDGETS.

（1）打开 envi.men 文件并在 MyFunctions 菜单下加入以下项目：

1 {Test Widgets} {not used} {test_widget}

（2）保存菜单文件

（3）重新启动 ENVI。在运行 ENVI 时段的 IDLDE 窗口中，打开一个新的编辑窗口：

（4）在新的编辑窗口中，输入下面的用户函数代码：

```
pro TEST_WIDGETS,event

compile_opt strictarr

tlb=widget_auto_base(title='widget test')
row_base1=widget_base(tlb,/row)
p1=widget_param(row_base1,/auto_manage,dt=4,field=2,$
                prompt='enter the first parameter',uvalue='p1')

row_base2=widget_base(tlb,/row)
p2=widget_param(row_base2,/auto_manage,dt=4,field=2,$
                prompt='enter the second parameter',uvalue='p2')
row_base3=widget_base(tlb,/row)
operation=widget_toggle(row_base3,/auto_manage,default=0,$
                    list=['add','multiply'],prompt='operation',$
                    uvalue='operation')
result=auto_wid_mng(tlb)

if(result.accept eq 0) then return

if(result.operation eq 0) then $
    envi_info_wid,strtrim(result.p1+result.p2) else $
        envi_info_wid,strtrim(result.p1*result.p2)

end
```

（5）将程序保存并存放到 ENVI 安装路径下的 save_add 目录下。

（6）通过 ENVI 菜单启动用户函数，查看结果。

七、用户函数中的错误捕获

如果可能的话，强烈建议对用户函数运行时可能发生的错误进行捕获以防止整个 ENVI 程序的崩溃。当然考虑到所有可能的情况是不可能的。这里介绍一些快速、简单的方法来避免最常见的错误。简单描述如下：

如果一个部件包括 Cancel 按钮，一定要检查看 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。

1、I/O 错误处理

尽管不是必须的，但是强烈推荐在处理程序中正确的处理 I/O 错误。这个工作可以通过 IDL 函数 ON_IOERROR 来完成。当 I/O 错误发生的时候，ON_IOEEROR 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，而错误信息存储在!ERROR_STATE.MSG 变量中。

此外还有 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使用未定义的函数。

ENVI 提供了 ENVI_IO_ERROR 函数来显示错误信息，使用这个函数进行用户函数的错误显示并和 ENVI 系统的错误显示界面保持一致。

2、例子：I/O 错误处理

本例说明了使用 ON_IOERROR 和 ENVI_IO_ERROR 进行错误捕获和错误信息显示的过程。强烈推荐所有的处理程序都进行 I/O 错误的检测处理和并显示相关的错误信息。下面的步骤列出了在处理程序中检测 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

3、使用 Catch 函数进行非 I/O 错误的异常捕获

CATCH 是一个特定的 IDL 程序，允许用户使用 IDL 内部的错误捕获机制来进行常用的错误处理以阻止异常错误的发生。IDL 的系统变量!ERROR_STATE 包含了最新的错误信息，包括内部的错误代码以及错误的信息。每次 IDL 的错误发生，整个系统变量被更新。但 CATCH 语句被调用后，ERROR 变量的值设置为 0。然后当一个错误发生是，ERROR 被设置为内部的错误代码，IDL 程序将会立即跳转到 CATCH 语句并从那儿开始执行。因此，您能够使用存储在!ERROR_STATE 系统变量中的错误信息，显示错误消息，并提示用户选择如何进行下一步的处理。

下面的例子说明了如何使用 CATCH 进行异常的捕获：

```
Catch, error
IF (error NE 0) THEN BEGIN
ok = DIALOG_MESSAGE(!error_state.msg, /cancel)
IF (STRUPCASE(ok) EQ 'CANCEL') THEN return
ENDIF
```

八、与显示窗口进行交互

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。一旦获得某一显示组的 DN 值，就可以使用 ENVI 提供的几个函数，获取显示组的有用信息，并能够控制 Zoom 窗口的移动位置。

ENVI_DISP_QUERY

该函数体能够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（ZOOM 窗口的中心位置）。

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。

九、使用影像分块技术

1、影像分块简介

ENVI 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。通常 ENVI 处理函数都是和影像分块技术结合的，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。

所有的 ENVI 用户函数也能够通过 ENVI 内建的分块函数来获取数据。这确保了用户函数也能够处理任意大小的数据文件。ENVI 的分块有三种格式：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，但是不推荐用户使用，因为它仅能用于比较小的文件。当然它是一种快速的数据访问方法，因此可用于快速进行程序原型的开发。

2、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，可以是空间方式划分也可以是波谱方式划分，以保证所有大小的影像都能被处理。一个空间分块的大小是 nlines*Sample，而波谱分块的大小是 Sample*band。

空间分块和波谱分块大小如下图所示：

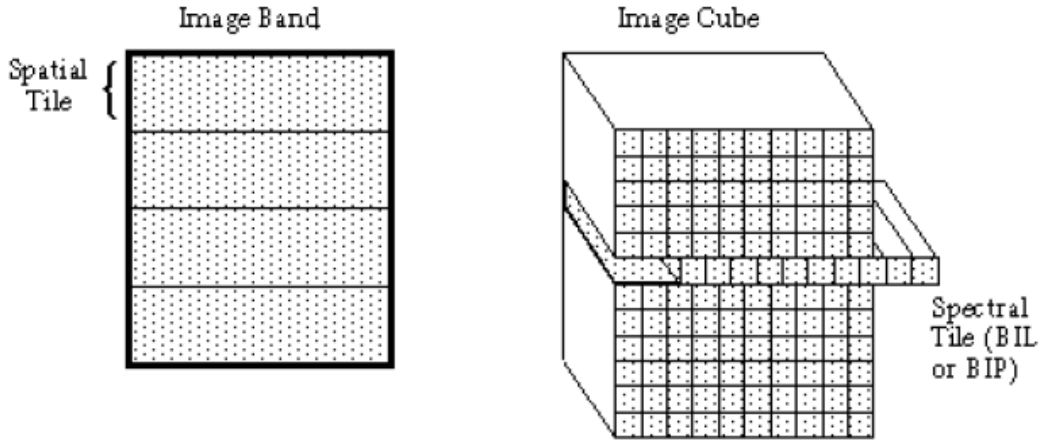


Figure 4-19: Illustration of Spatial and Spectral Tiles

空间分块近似于按输入波段对影像进行分块，因此可以进行空间处理而不用考虑文件的存储方式。当文件中具有多个波段，所有波段都拥有同样数目的空间分块。通常对 BSQ 文件进行空间分块，而对 BIL 或 BIP 文件进行波谱分块效率非常高。

在进行邻域处理时，空间分块也可以设定重叠的行数。重叠行加在每个分块的顶部，当整个波段作为一个分块时，不起增加重叠行。例如：进行 3x3 卷积时，需要重叠一行来对上一个分块的最后一行进行处理。

分块处理的步骤如下：

①、初始化空间或波谱分块需求，使用 ENVI_INIT_TILE

②、获取分块输入数据，ENVI_GET_TILE

③、当所有的分块数据都处理完毕，释放分块需求,ENVI_TILE_DONE

3、分块处理程序例子

例 1：空间分块程序示例

空间分块的处理程序不需要担心输入数据的存储顺序。分块初始化函数需要一个文件 ID，所需波段以及空间纬度。在用 ENVI_SELECT 函数来选择输入文件时，该函数返回所有这些参数。

本例中使用 ENVI_SELECT 来选择输入文件，返回 FID、POS 以及 DIMS 变量值并作为 ENVI_INIT_TILE 的输入值。

ENVI_INIT_TILE 返回的 TILE_ID 是与需求分块联系的唯一参考 ID，并可以被其它分块函数所使用。通过 FOR 循环来处理所有数目的分块。在 FOR 循环中，当前的分块，波段索引、以及分块数据信息将被打印到 IDL 的日志窗口。一旦所有的分块被处理完成，通过使用 ENVI_TILE_DONE 释放分块需求。

示例如下：

```
;本例对影像进行分块，并打印出每块数据的信息
pro spat_tile
;首先选取一个文件
envi_select, title=’Input Filename’, fid=fid, $
pos=pos, dims=dims
;如果打开文件失败则返回
if (fid eq -1) then return
;获取分块需求，返回 tile_id 和分块数 num_tiles
tile_id = envi_init_tile(fid, pos, interleave=0, $
num_tiles=num_tiles, xs=dims[1], xe=dims[2], $
ys=dims[3], ye=dims[4])
;对每个分块进行处理
for i=0L, num_tiles-1 do begin
data = envi_get_tile(tile_id, i, band_index=band_index)
print, i, band_index
help, data
endfor
;释放分块需求
envi_tile_done, tile_id
```

end

通过下列步骤执行本例：

（1） 保存该函数到文件并将其放置到 save_add 目录下

（2） 启动或重新启动 ENVI

（3） 打开 IDL 开发环境（PC）或是 ENVI 的 shell 窗口（UNIX、MAC OSX）

（4） 在 ENVI 的命令提示行下输入：

Spat_tile

（5）选择一个输入文件，并查看 IDL 日志窗口中的显示的信息。

例 2：波谱分块示例

使用波谱分块处理需要考虑到 BIL 和 BIP 的存储方式以优化性能。分块初始化程序需要文件 ID、所选的波段数、影像大小。当使用 ENVI_SELECT 函数来选择输入文件时，该函数返回了这些参数。

本例中的处理程序使用 ENVI_SELECT 来选取输入文件，返回所选的 FID、POS、以及 DIMS 变量并输入给 ENVI_INIT_TILE 函数。本例中使用 BIL 的分块方式对 BSQ 或 BIL 方式存储的数据，而对 BIP 的数据采用 BIP 的分块方式。输入数据的存储方式可以使用 ENVI_FILE_QUERY 函数的 INTERLEAVE 关键字获取。

本例代码如下：

```
;波谱分块例子
;对输入的数据进行波谱分块并打印每一块数据
pro spec_tile

;选择文件
envi_select,title='Input FileName',fid=fid,$
    pos=pos,dims=dims
;如果文件打开错误，返回
if (fid eq -1) then return

;获取文件的存储方式
envi_file_query, fid, interleave=interleave

;设置波谱分块需求
;tile_id 是和分块需求相关联的参考 ID，其它函数可以 tile_id 访问分块需求
tile_id = envi_init_tile(fid, pos, num_tiles=num_tiles, $
interleave=interleave>1, xs=dims[1], xe=dims[2], $
ys=dims[3], ye=dims[4])
```

```
;对每块数据进行处理
for i=0L, num_tiles-1 do begin
data = envi_get_tile(tile_id, i)
print, i
help, data
endfor

;释放分块需求
envi_tile_done, tile_id

end
```

通过下列步骤执行本例：

（5） 保存该函数到文件并将其放置到 save_add 目录下

（6） 启动或重新启动 ENVI

（7） 打开 IDL 开发环境（PC）或是 ENVI 的 shell 窗口（UNIX、MAC OSX）

（8） 在 ENVI 的命令提示行下输入：

Spec_tile

（5）选择一个输入文件，并查看 IDL 日志窗口中的显示的信息。

例 3：根据输入文件存储方式自动确定空间分块还是波谱分块

通过设定将分块程序的 interleave 关键字设置为输入数据文件的 interleave 值，可以实现根据数据的存储方式自动设定分块方式。

示例代码如下：

```
;本例说明针对输入文件的存储类型进行不同的分块
;由 Interleave 控制分块的类型
;0(BSQ)空间分块
;1(BIL),2(BIP)波谱分块

pro ss_tile
;选择文件
envi_select,title='Input FileName',fid=fid,$
   pos=pos,dims=dims
;如果文件打开失败则退出程序
if (fid eq -1) then return
```

```
;获取文件的存储方式
envi_file_query, fid, interleave=interleave
;设定分块需求
tile_id = envi_init_tile(fid, pos, num_tiles=num_tiles, $
interleave=interleave, xs=dims[1], xe=dims[2], $
ys=dims[3], ye=dims[4])

;获取分块数据并对数据进行处理
for i=0L, num_tiles-1 do begin
data = envi_get_tile(tile_id, i)
; Sample case statement for BSQ, BIL and BIP tiles
case interleave of
0:
1:
2:
endcase
print, i
help, data
endfor

;释放分块需求
envi_tile_done, tile_id
end
```

通过下列步骤执行本例：

（9） 保存该函数到文件并将其放置到 save_add 目录下

（10） 启动或重新启动 ENVI

（11） 打开 IDL 开发环境（PC）或是 ENVI 的 shell 窗口（UNIX、MAC OSX）

（12） 在 ENVI 的命令提示行下输入：

ss_tile

（5）选择一个输入文件，并查看 IDL 日志窗口中的显示的信息。

4、保存结果

处理结果能够被写入文件或是存入内存，当影像结果被写入文件时，建议使用分块所使用的存储方式。这样空间分块 BSQ 的结果将产生 BSQ 格式的结果，同样采用 BIP 分块产生的结果将产生 BIP 的结果。这样处理后的分块就可以直接写入文件而无需转换存储方式。

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。下列文件信息必须要写入头文件：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，结果存储在内存分配的数组中。处理后的数据块插入到合适的存储位置。内存数组的大小为NS*NL*NB，IDL数组创建函数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。

当处理完成后，存储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，仅仅需要提供内存数组。但有些时候也需要提供一些额外的信息，如X、Y的起始位置、文字描述和波段名称。

实例：空间分块结果存入硬盘

本例将前面空间分块处理的程序进行了修改，经分块处理后的结果存入到硬盘并写入一个ENVI头文件。修改的部分如下：

（1） 接受一个输出文件名作为参数；

（2） 打开输出文件并写入结果；

（3） 调用ENVI_SETUP_HEAD函数写入ENVI头文件。

IDL的OPENW程序用来打开输入的文件名，并使用GET_LUN关键字自动获取一个文件单元号。在分块处理循环内部，每次处理完的分块数据使用IDL的WRITEU程序写入硬盘。WRTITEU使用OPENW返回的文件单元号和分块数据作为参数。采用与分块数据同样的数据类型采用二进制流的方式写入文件。由于实际上没有进行任何的处理，输出的数据类型和输入的数据类型是同样的。

注意：在分块处理过程中，数据类型可能会发生改变，因此要检查输出数据的类型是否是用户所需的。

在把所有的数据写入到输出文件后，关闭文件并使用IDL的FREE_LUN函数释放相关联的文件单元号。并使用ENVI_SETUP_HEAD写入ENVI头文件。

示例代码如下：

```
;将空间分块处理结果存入文件
;增加一个输入参数，为输出的文件名

pro spat_disk, out_name
;检查 out_name 参数是否设置
```

```
if (n_elements(out_name) eq 0) then begin
print, ’Please specify a valid output filename’
return
endif

;选择输入文件
envi_select,title='Input Filename',fid=fid,$
   pos=pos,dims=dims

;如果文件打开失败退出程序
if (fid eq -1) then return

;查询文件信息
envi_file_query, fid, data_type=data_type, xstart=xstart,$
ystart=ystart
ns = dims[2] - dims[1] + 1
nl = dims[4] - dims[3] + 1
nb = n_elements(pos)

;打开文件
openw, unit, out_name, /get_lun

;设置分块需求
tile_id = envi_init_tile(fid, pos, interleave=0, $
num_tiles=num_tiles, xs=dims[1], xe=dims[2], $
ys=dims[3], ye=dims[4])

;对分块数据进行处理
for i=0L, num_tiles-1 do begin
data = envi_get_tile(tile_id, i, band_index=band_index)
print, i, band_index

;将数据写入文件
writeu, unit, data
endfor
```

```
;释放文件单元号
free_lun, unit

;写入 ENVI 头文件
envi_setup_head, fname=out_name, ns=ns, nl=nl, nb=nb, $
data_type=data_type, offset=0, interleave=0, $
xstart=xstart+dims[1], ystart=ystart+dims[3], $
descrip='Test routine output', /write, /open

;释放分块需求资源
envi_tile_done, tile_id
end
```

执行下面的步骤执行本例：

（1） 保存程序到文件并将其放置的 save_add 目录下

（2） 启动或重启 ENVI

（3） 打开 IDL 开发环境(PC Macintosh) 或者是 ENVI 的 Shell 窗口（UNIX）

（4） 在 ENVI 的命令行下输入下面命令： spat_disk,'filename'

（5） 选择一个输入文件

（6） 结果影像将被加入到可用波段列表

实例：将分块处理结果存入内存

本例是将上面空间分块结果保存到磁盘进行改动，将结果保存到内存中。数据使用 ENVI_ENTER_DATA 输入到 ENVI 中。

IDL 程序 MAKE_ARRAY 用来为动态分配输出内存数组。在本例中，通过 TYPE 关键字设置输出数组的数据类型。

示例代码如下：

```
    ;本例将分块结果处理后存入内存
   ;本例需要启动一个交互式的 ENVI
;

pro spat_mem

;选择文件
```

```
envi_select,title='Input FileName',fid=fid,$
    pos=pos,dims=dims

;如果打开文件失败，退出程序
if (fid eq -1) then return

;查询文件信息
envi_file_query, fid, data_type=data_type, xstart=xstart,$
ystart=ystart
ns = dims[2] - dims[1] + 1
nl = dims[4] - dims[3] + 1
nb = n_elements(pos)

;构建内存数组
mem_res = make_array(ns, nl, nb, type=data_type, /nozero)

;设定分块需求
tile_id = envi_init_tile(fid, pos, interleave=0, $
num_tiles=num_tiles, xs=dims[1], xe=dims[2], $
ys=dims[3], ye=dims[4])

;对每个分块数据进行处理
for i=0L, num_tiles-1 do begin
data = envi_get_tile(tile_id, i, band_index=band_index, $
ys=ys)
print, i, band_index

;将分块数据存入内存数组适当位置
mem_res[0,ys-dims[3],band_index] = data
endfor

;将内存数据送入 ENVI，显示在可用波段列表中
envi_enter_data, mem_res, xstart=xstart+dims[1], $
ystart=ystart+dims[3], descrip='Test routine output'
```

```
;释放分块资源
envi_tile_done, tile_id
end
```

通过下面的步骤执行本例：

（1） 将程序保存到文件并放置到 save_add 目录下

（2） 启动或是重新启动 ENVI

（3） 在 IDL 的开发环境或是 ENVI 的 shell 窗口中输入下面命令：spat_mem

（4） 选择一个输入文件

（5） 结果影像加入到可用波段列表中

实例：将波谱分块结果保存到磁盘

```
pro spec_disk,out_name

  ;Check for an output filename

  if(n_elements(out_name) eq 0) then begin
      print,'Please specify a valid output filename'
      return
  endif
  envi_select,title='Input Filename',fid=fid,$
      pos=pos,dims=dims

  if(fid eq -1) then return
  envi_file_query,fid,data_type=data_type,xstart=xstart,$
     ystart=ystart,interleave=interleave

  ns=dims[2]-dims[1]+1
  nl=dims[4]-dims[3]+1
  nb=n_elements(pos)

  openw,unit,out_name,/get_lun
  tile_id=envi_init_tile(fid,pos,num_tiles=num_tiles,$
       interleave=(interleave>1),xs=dims[1],xe=dims[2],$
       ys=dims[3],ye=dims[4])

  for i=0L,num_tiles-1 do begin
```

```
        data=envi_get_tile(tile_id,i)
        writeu,unit,data
        print,i
    endfor

    free_lun,unit
    envi_setup_head,fname=out_name,ns=ns,nl=nl,nb=nb,$
       data_type=data_type,offset=0,interleave=(interleave>1),$
       xstart=xstart+dims[1],ystart=ystart+dims[3],$
       descrip='Test routine output',/write,/open

    envi_tile_done,tile_id


end
```

通过下面的步骤执行本例：

（1） 将程序保存到文件并放置到 save_add 目录下

（2） 启动或是重新启动 ENVI

（3） 在 IDL 的开发环境或是 ENVI 的 shell 窗口中输入下面命令：spec_disk,’filename’

（4） 选择一个输入文件

（5） 结果影像加入到可用波段列表中

5、非分块处理程序

ENVI 也提供了非分块的处理方法。对于分块处理来说，输入数据以空间或是以波谱的方式被平分为同样大小。而非分块处理程序一次获取整个波段或整个空间子集的数据。ENVI 不会对获取数据的大小进行限制。

ENVI 提供了两个函数用来获取非分块数据，功能如下：

ENVI_GET_DATA 获取空间数据（一次只能获取一个波段数据）

ENVI_GET_SLICE 获取整个波谱数据

例子：非分块空间处理

本例获取一个波段的空间数据，使用 ENVI_SELECT 用来选择输入数据。

（1）启动 ENVI

（2）打开一个多波段影像

（3）打开 IDL 开发环境或是 ENVI shell 窗口

（4）在 ENVI 命令行下输入以下命令：

Envi_select,title='Input Filename', fid=fid,pos=pos,dims=dims

（5）获取第一个波段数据：

Data=envi_get_data(fid=fid,dims=dims,pos=pos[0])

（6）确认获取的数据：help,data

（7）可以设置 POS 的值来获取其它波段的数据：

Data=envi_get_data(fid=fid,dims=dims,pos=pos[1])

例子：非分块波谱处理

本例交互式的获取一个波谱数据块。

（1）启动 ENVI

（2）打开一个多波段的影像

（3）打开 IDL 开发环境或 ENVI 的 shell 窗口

（4）在 ENVI 的命令行中输入以下命令：

Envi_select,title='Input Filename',fid=fid,pos=pos,$

Dims=dims

（5）获取第一行的 BIL 数据切片

Data=envi_get_slice(fid=fid,pos=pos,line=dims[3],$

Xs=dims[1],xe=dims[2],/bil)

（6）验证数据：help，data

（7）获取最后一行的 BIP 数据切片

Data=envi_get_slice(fid=fid,pos=pos,line=dims[4],$

Xs=dims[1],xe=dims[2],/bip)

6、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 提供的进度报告工具，开发人员只需控制增量大小和更新频率就能够实现进度报告。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 命令将被忽略。

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

1、 启动 ENVI

2、 在 IDL 命令行中键入以下命令：

3、 初始化进度对话框：

```
Envi_report_init,['Input File:filename',$
        'Output File:filename'],title='Test Status',base=base
```

4、 设置进度增量

```
ENVI_REPORT_INC,base,3
```

5、 进行进度更新：

```
ENVI_REPORT_STAT,base,1,3
ENVI_REPORT_STAT,base,2,3
ENVI_REPORT_STAT,base,3,3
```

6、 完成进度对话框

```
ENVI_REPORT_INIT,base=base,/finish
```

十、对于 ENVI 用户函数有用的 IDL 函数

FORWARD_FUNCTION 或 COMPILE_OPT STRICTARR

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，那时变量名是函数名，而非数组定义。而 COMPILE_OPT STRICTARR 则强制编译器以[]作为数组的定义。

RESOLVE_ALL

在 IDL 程序中用到的许多 IDL 内置的函数，都是以源码的形式提供的。在 IDL 编译器中，它们能够被自动编译。但是在 ENVI 环境中，ENVI 不能编译这些函数，因此要将用户函数打包，必须要找到所有依赖的 IDL 函数，而 IDL 编译器提供了一个工具就是 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：编译一个 SAVE 文件

1、 在用户函数中加入 COMPILE_OPT STRICTARR 或是使用 FORWARD_FUNCTION

函数

2、 保存修改后的代码

3、 启动一个新的 Session

4、 编译修改后的用户函数

5、 使用 RESOLVE_ALL 函数编译所有依赖函数：

RESOLVE_ALL, /CONTINUE_ON_ERROR

6、 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。

第五章 常用编程工具

一、绘图

ENVI 为用户提供了访问 ENVI 绘图窗口的方法。用户能够绘制一系列的图形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，轴名以及绘图的标题。

例子：

绘制数据

1、 启动 ENVI

2、 在 IDL 命令行中输入以下命令

```
X=findgen(100)
Y=randomu(seed,100)
Envi_plot_data,x,y,plot_title='Numbers',$
  Plot_colors=[10],plot_names='Random',$
  Xtitle='Count',ytitle='Value'
```

二、报告

用户可以使用 ENVI 提供的报告部件来显示文本信息。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作为一行显示。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

1、 启动 ENVI

2、 在 IDL 命令行下输入以下命令：

```
Str=['Line 1','Next line is blank','','Line 4']
ENVI_INFO_WID,str, title='Report'
```

三、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，函数内部采用了取模运算。

注：用户可以通过修改 ENVI 目录下的 colors.txt 文件，添加自定义的颜色到 ENVI 系

统中。

例子：获取 RGB 颜色值

1、 启动 ENVI

2、 在 IDL 命令行中输入以下命令：

```
Envi_get_rgb_triplets,0,r,g,b
Print,r,g,b

Envi_get_rgb,triplets,1,r,g,b
Print,r,g,b
```

四、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

1、 启动 ENVI

2、 在 IDL 命令下输入以下命令：

```
Envi_select, title='Input Filename' ,fid=fid

Envi_file_query,fid, ns=ns, nl=nl , nb=nb, offset=offset,$
    Data_type=data_type,interleave=interleave

Print,ns,nl,nb,offset,data_type,interleave
```

例子：获取地图信息

1、 启动 ENVI

2、 在 IDL 命令行中输入以下命令：

```
Envi_select,title='Georeferenced  Filename',fid=fid

Map_info=ENVI_GET_MAP_INFO(FID=fid)

Proj=ENVI_GET_MAP_INFO(FID=fid)

Print,proj
```

五、文件的管理

ENVI 提供了几个工具用于打开、关闭以及选择影像文件，这些工具可用用于交互或是批处理模式。交互 ENVI 模式下使用 ENVI_SELECT 选择 ENVI 影像文件，ENVI_SELECT 函数返回 FID，作为对文件的引用。在选择非影像文件如 ROI 文件时，使用 ENVI_PICKFILE 获取文件名。在非影像的情形下，文件名提供链接就能够提取出必须的信息。例如，ENVI_FILE_QUERY 使用 FID 引用文件，而 ENVI_RESTORE_ROIS 使用文件名来恢复感兴趣区。

在进行批处理编程时，使用 ENVI_OPEN_FILE 可以打开任意 ENVI 文件并返回 FID 而无需用户交互。批处理和交互模式下编程都可以使用 ENVI_FILE_MNG 来关闭和删除影像文件。

下面列出了用于进行文件管理的 ENVI 函数：

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

例子：交换式选择文件

本例用 ENVI_SELECT 选择一个文件，接着使用 ENVI_FILE_MNG 关闭该文件。启动 ENVI，在命令行输入 ENVI_SELECT。一但文件选择对话框出现，可以选择一个打开的文件，或是打开一个新的影像文件。一但选择了影像文件点击 OK，返回的 FID 将被 ENVI_FILE_MNG 函数用于关闭该文件。

如果用户点击了 ENVI_SELECT 对话框中的 Cancel 按钮，FID 将返回为-1，表示没有文件被选择。

1、 启动 ENVI

2、 打开 IDL 开发环境 IDLDE（PC）或是着 ENVI 启动的 shell 窗口（UNIX，Mac OS X）

3、 在 ENVI 的命令行输入以下命令选择文件：

envi_select, title=’Input FileName’,fid=fid

4、 在 ENVI 的命令行中输入以下命令关闭该文件：

5、 Envi_file_mng, id=fid,/remove

第六章 交互式用户函数

一、绘图函数

绘图函数能够将各种变换或是方程数据绘制在绘图窗口。当选择了不同的绘图函数，数据将被再次输入函数最终结果将显示在绘图窗口中。新的数据必须首先应用方程，然后再进行显示。通常绘图函数用于绘制 Z 剖面的光谱数据，光谱库，ROI 均值，或是其它的数据，但数据源对这些函数来说不是必需的。

ENVI 提供了标准设置的绘图函数，并允许用户自定义绘图函数。自定义的绘图函数接受输入数据，应用处理，并输出转换后结果到绘图窗口。任何数据都可以通过选择绘图函数进行绘制。绘图函数仅对 Y 轴数据操作，而 X 轴数据保持不变。

自定义的绘图函数可以通过修改 ENVI 菜单以及将函数名称加入到 menu 目录下的useradd.txt 文件中。在 useradd.txt 文件中，绘图函数使用{plot}标识区别于其它的函数。通常的绘图函数的格式如下：

{plot}{Button Name}{function_name}{type=n} 这里：

{plot}表示该行是绘图函数的定义

{Button Name} 绘图窗口下拉菜单中的相对应菜单名称

{function_name}调用的绘图函数名

{type=n} 绘图函数的更新类型。设置 type=0 当新数据可用的时候调用绘图函数，设置 type=1 当新数据可用或是绘图窗口缩放时调用绘图函数。

Useradd.txt 文件示例加入用户自定义的绘图函数：

{plot} {Normal} {sp_normal} {type=0}

{plot} {Continuum Removed} {sp_continuum_removed} {type=1}

{plot} {Binary Encoding} {sp_binary_encoding} {type=0}

要加入一个自定义的绘图函数并且仅在新数据的时候更新，在 Useradd.txt 文件中加入下列内容：

{plot} {My function} {my_plot_func} {type=0}

改变前面自定义函数的设置，使它在新数据和数据缩放的时候都更新，改变 type 的值为 1，如下所示：

{plot} {My function} {my_plot_func } {type=1}

绘图函数定义时需要一些参数，如 X 和 Y 数据，坏波段信息，以及左、右方向缩放的值。ENVI 在调用绘图函数时，会自动传递这些必需的参数。下面列出了绘图函数定义的一个例子：

```
FUNCTION my_func , X, Y, BBL ,BBL_ARRAY, L_POS=L_POS, $
 R_POS=R_POS, _EXTRA=_EXTRA
```

这里：

My_func 为绘图函数名

X X 轴数据名

Y Y 轴数据名

BBL Z 剖面中每个坏波段的指针，如果没有坏波段或是当前绘图不是 Z 剖面，该值为未定义。

BBL_ARRAY –由 0 和 1 组成的数据代表绘图数据中坏和好的数据。BBL_ARRAY 的大小和 X 的大小一致。对所有的绘图都可以使用该数组而不用考虑数据的类型。

L_POS –X 数组的最小索引值

R_POS –X 数组的最大索引值

EXTRA— 必须定义的关键字，以收集 ENVI 传递的其它关键字

新的 Y 轴数据从绘图函数中返回，下面是一个绘图函数的模板：

```
FUNCTION my_plot_func, X, Y, BBL, BBL_ARRAY, $
    L_POS=L_POS, R_POS=R_POS, _EXTRA=_EXTRA
    Statements….
    RETURN, plot_result
END
```

绘图函数在处理 Z 轴剖面数据时可能要涉及到传递给函数坏波段信息。坏波段在绘图函数计算是会忽略，它们的输出值将不会显示在绘图窗口上。

为了忽略坏波段值，定义一个指针指向好的波段并将其做为 X 和 Y 数组的索引。下面的语句设置了 PTR 变量指向好的数据点的索引而 COUNT 变量等于好的点的数目。

```
Ptr=where(bbl_array eq 1, count)
```

下面的例子近一步说明了绘图函数的实现：

例子：绘图函数

本例编写一个绘图函数用于计算绘图的零值平均。该函数仅计算绘图中好的数据点，并且当绘图缩放时绘图数据不会变化。首先，通过检查 BBL_ARRAY 获取好的数据点，并构建一个索引使所有好的点设置为 1。接着将好的数据点从 Y 轴数据中抽取出来计算均值。如果没有合法的数据点存在，结果设置为 0，从函数中返回 Y 轴数据结果。

示例代码如下：

```
Function pf_zero_mean, x, y, bbl, bbl_array, _extra=_extra
    Bbl_ptr=where(bbl_array eq 1,count)
    If(count gt 0) then $
        Result=y-(total(y[bbl_ptr])/count) $
    Else $
      Result=fltarr(n_elements(y))
    Return, result
```

End

将这个函数保存为文件，并将它放置到 ENVI 安装目录下的 save_add 目录下。此外，下面的内容必须加入到 ENVI 安装目录下的 useradd.txt 文件中，这能够在绘图窗口中的下拉菜单下访问该函数。

{plot} {Zero Mean} {pf_zero_mean} {type=0}

下面列出了执行这个函数所需的步骤：

1、 将该函数保存为文件并将它放置到 save_add 目录下

2、 添加绘制函数定义到 useradd.txt

3、 启动 ENVI

4、 打开一个文件并显示 X 剖面数据

5、 在绘图窗口下拉菜单下选择 Zero Mean。

二、波谱分析函数

波谱分析函数用来匹配未知波谱和波谱库中物质的相似程度。ENVI 包括了常用的波谱区分技术，如二进制编码、波谱角制图以及波谱特性匹配。同时用户自定义的函数也能够加入到波谱分析中并和 ENVI 中已提供的函数一起使用。每个函数都获得的 0 到 1 的得分，结果以匹配等级最好到最坏的顺序进行排列。

自定义的波谱分析函数必须要在 useradd.txt 文件中加入菜单名和函数名才能显示在 ENVI 的相应菜单中。定义时使用{identify}标识来区分波谱分析函数。定义的格式为：

{identify} {Method Name} {Out Name} {func_name} {min, max}

这里：

{identify}—标识了波谱分析函数的定义

{Method Name} – 波谱识别部件的方法名

{Out Name} – 波谱分析输出等级报告窗口的列名

{func_name}—波谱识别函数调用名称。

{min, max}—识别函数的默认最小和最大输出。默认的最小值和最大值可以在允许波谱分析时进行编辑。当前的缩放系数也被传递给识别函数，输出结果将被缩放到[0,1]之间以进行所有方法的累积和计算。

波谱分析函数由两部分组成。第一部分为设置程序，它在选择了指定波谱库后被调用。所有不依赖于输入数据的库运算都在该步处理中执行。该程序具有附加名“_SETUP”。例如，FUNC_NAME 函数的 setup 程序名为 FUNC_NAME_SETUP。

(注：不同于识别函数，setup 必须定义为程序，并且必须定义即使不需要 Setup 步骤。)

FUNC_NAME 函数的 Setup 程序定义如下：

Pro func_name_setup, WL, SPEC_LIB, HANDLES, NUM_SPEC=NUM_SPEC

这里：

WL--- 波谱库的波长值。库中的所有波谱都在单一波长进行一次采样，这样对于每一个库就有一个单一的波长矢量库。

SPEC_LIB　所有波谱库波谱的二维数组。维数为[波长采样数，波谱数]

HANDLES　用来存储用户数据的数组。任何预处理的波谱库要被存储在这些处理程序中。这个处理数组将传递给识别函数。

NUM_SPEC　库中的波谱数。该值等于 SPEC_LIB 数组的第二维大小。

Setup 程序和识别函数组合起来就构成了波谱分析函数。结果的分值基于当前的权重自动的和其它波谱分析函数合并。

下面的例子说明了一个波谱分析函数的实现方法。

示例：波谱分析函数

这个例子创建一个波谱分析函数用于计算当前波谱和波谱库中每一个波谱的最小距离。Setup 程序虽被定义但是空的，因为本例中无需 Setup 处理。识别函数以距离差之和的平方根形式计算每个波长的距离。同时每个波谱库的波谱和输入参考波谱都计算一个距离权重值。输出的权重值以最大误差为序排列，以保持距离测量的值在 0 到 1 之间。

注：本函数仅是示例。真正的最小距离波谱分析函数在计算最小距离前需要先移除波谱库和参考波谱中的连续性。

示例的代码如下：

```
Pro isadist_func_setup, w1, spec_lib, handles, num_spec=num_spec
    ; No initialization is necessary
end
function irsadist_func, w1, ref, spec_lib, handles, num_spec=num_spec, $
          scale_vals
    ; Compute the distance compared to each library member
    Result=dblarr(num_spec)
    For i=0L,num_spec-1 do $
      Result[i]=sqrt(total((spec_lib[*,i]-ref)^2,/double))
    Dmax=max(result,min=dmin)
    Return, (1d-((result-dmin)/(dmax-dmin)))/scale_vals(1)
end
```

将该程序保存到文件，并将其放置在 save_add 目录下。在 useradd.txt 文件中加入下列内容，允许在波谱分析中选择该函数。

```
{identify} {Minimum Distance} {MDIST} {irsadist_func} {0,1.}
```

通过下面的步骤执行这个例子：

1、 保存该函数到文件并将它存放到 save_add 目录下；

2、 将波谱函数定义加入到 useradd.txt 文件中；

3、 启动 ENVI；

4、 打开一个文件并显示 Z 剖面；

5、 打开波谱工具下的波谱分析，并打开一个波谱库进行比较；

6、 设置最小距离权重为 1.0 并选择 OK；

7、 选择 Apply 给当前的 Z 剖面曲线进行分级。

三、用户定义的地图投影类型

ENVI 支持许多不同的地图投影以及地图投影类型。用户也能够创建自定义的地图投影并使用 ENVI 主菜单下的 Map-Customize Map Projections 选取已有的地图投影类型。然而用户有时需要定义自己的投影类型。这时，用户通过编写 IDL 程序并计算转换前经纬度和转换后新的投影坐标系统中的坐标来定义新的投影类型，并能够将其加入到 ENVI 中。

为了定义新的投影：

1、 从 ENVI 主菜单中，选择 Map- Customize Map Projections；

2、 从投影类型中选择新定义的投影类型；

3、 输入投影参数。

用户自定义的投影函数需要先加入到 map_proj.txt 中。在 useradd.txt 中加入自定义的投影类型并输入投影名称以及投影函数名。地图投影函数采用{projection type}标识区别于其它的函数。定义的格式如下：

{projection type} {projection name} {routine_root_name} {number of extra parameters}

这里：

{projection type} – 标识用户定义的投影类型

{projection name}－列在投影类型中的投影名称

{routine_root_name}－用户函数的名称

{number of extra parameters}－该投影所需的附加参数数目：椭球体参数 a, b； 投影的起始点经纬度；东、北偏移。如果无需其它参数，该值可以设置为{0}，该值定义的最大值是{9}。

用户定义的投影函数由两部分组成，第一部分允许用户输入额外的参数，第二部分进行坐标转换。如果需要输入额外的参数，一个名为”routine_root_name”_DEFINE 函数用于输入这些参数。该函数有一个 DBLARR 类型的数组保存额外的参数。这些额外的参数是双精度型的浮点数，最多允许输入 9 个。定义程序通过构建投影对话框或是通过在程序中设置来分配这些额外的参数值。（如果额外参数的个数为{0}，那么无需 routine_root_name 程序。）

第二部分需要命名为 routine_root_name_CONVERT。它执行经纬度和新的投影坐标的相互转换。该程序有 6 个参数，如下所示：

Routine_root_name_convert,x,y,lat,lon,to_map_=to_map,projection=proj

这里：

X, y －地图投影的坐标

Lat/lon－经纬度坐标

TO_MAP－如果设置了该参数，程序将经纬度转换为地图坐标 X,Y；如果没有设置，程序从地图坐标 X,Y 转换为经纬度。

PROJ－用户定义的{envi_proj_struct}参数

其中：

Proj.type=99

Proj.params[0]=a　－椭球体参数 a

Proj.params[1]=b　－椭球体参数 b

Proj.params[2]=lat0 －投影原点的纬度

Proj.params[3]=lon0 －投影原点的经度

Proj.params[4]=x0　－东偏移

Proj.params[5]=y0　－北偏移

Proj.params[6:*]=　附加的用户参数

将该函数的.pro 或..sav 文件加入到 ENVI 安装目录下 save_add 目录下。

为了完成投影定义需要执行以下步骤：

1、 启动 ENVI；

2、 在 ENVI 菜单中，选择 Map－Customize Map Projections；

3、 最新定义的投影类型将会显示在投影类型列表中；

4、 选择投影类型并输入参数值。

保存到 map_proj.txt 文件中的投影类型，这样所有的 ENVI 投影工具能够使用该投影。

例子：用户定义地图投影

下面两个例子说明了如何定义新的投影。USER_PROJ_TEST1 定义了无需额外参数的投影，USER_PROJ_TEST2 定义了需要额外参数的投影。根据用户实际定义投影类型的需要，用户可以以这两个例子作为参考。

例 1：USER_PROJ_TEST1

因为本例中没有额外的参数，因此只需_convert 程序。

本例代码如下：

```
Pro user_proj_test1_convert,x,y,lat,lon,to_map=to_map,projection=p
    If(keyword_set(to_map)) then begin
        X=lon
        Y=lat
    Endif else begin
        Lon=x
        Lat=y
    Endelse
```

End

保存该程序到文件并把它存放到 ENVI 安装目录下的 save_add 目录下。在 useradd.txt 文件中加入以下内容：

{projection type} {User Projection #1} {user_proj_test1} {0}

通过下面的步骤来执行这个例子：

1、 保存该函数并将其放置到 save_add 目录下；

2、 在 useradd.txt 文件中定义投影函数；

3、 启动 ENVI；

4、 通过 Map-Customize Map Projection 选择新的投影类型；

5、 正确输入参数后，用户将新的投影加入到 map_proj.txt 中。

例 2：USER_PROJ_TEST2

本例需要定义四个附加的参数，因此需要创建_define 以及_convert 函数，本例代码如下：

```
pro user_proj_test2_define,add_params
  if(n_elements(add_params) gt 0) then begin
     default_1=add_params[0]
     default_2=add_params[1]
     default_3=add_params[2]
     default_4=add_params[3]
  endif
   base=widget_auto_base(title='User Projection #1 Additional Parameters')
  sb=widget_base(base,/column,/frame)
  sb1=widget_base(sb,/row)
  wp=widget_param(sb1,prompt='Parameter #1',xsize=12,dt=4,$
     field=4,default=default_1,uvalue='param_1',/auto)
  sb1=widget_base(sb,/row)
  wp=widget_param(sb1,prompt='Parameter #2',xsize=12,dt=4,$
       field=4,default=default_2,uvalue='param_2',/auto)
  sb1=widget_base(sb,/row)
  wp=widget_param(sb1,prompt='Parameter #3',xsize=12,dt=4,$
       field=4,default=default_3,uvalue='param_3',/auto)
  sb1=widget_base(sb,/row)
  wp=widget_param(sb1,prompt='Parameter #4',xsize=12,dt=4,$
       field=4,default=default_4,uvalue='param_4',/auto)
  result=auto_wid_mng(base)
  if(result.accept) then $
```

```
    add_params=[result.param_1,result.param_2,$
                result.param_3,result.param_4]
end
  pro user_proj_test2_convert,x,y,lat,lon,to_map=to_map,$
        projection=p

   if(keyword_set(to_map)) then begin
     x=lon*100.+p.params[4]
     y=lat*100.+p.params[5]
   endif else begin
     lon=(x-p.params[4])/100.
     lat=(y-p.params[5])/100.
   endelse
end
```

保存该程序文件，并将其放到 save_add 目录下。此外要在 useradd.txt 文件中加入以下内容：

{projection type} {User Projection #2 } {user_proj_test2} {4}

通过执行下面的步骤，使用新的投影：

1、 保存文件到 save_add 目录下；

2、 在 useradd.txt 文件中加入新的投影定义；

3、 启动 ENVI；

4、 选择 Map- Customize Map Projections；

5、 从投影类型列表中选择新的投影类型；

6、 输入正确的参数后，可以将新建的投影存入 map_proj.txt 文件中。

四、用户自定义单位

ENVI 提供了不同的单位，能够在地图投影或是 ENVI 测量工具中选择。这些单位包括米、千米、英尺、码、英里、海里、英亩、公顷、度、分、秒以及弧度。ENVI 允许用户定义自己的单位并用在地图投影或是测量计算中。用户可以通过输入自定义单位与米、度、平方米相互转换的系数来定义该单位。

定义的用户单位可在 user_add.txt 文件中加入一个比例因子，这样该单位就可是在 ENVI 中使用。单位定义使用{units}来区分于 user_add.txt 文件中的其它函数。格式如下：

{units} {Name} {scale factors} {0|1|2}

这里：

{units}—用于表示单位定义

{scale factor}—将用户函数转换为米、度或平方米的比例因子

{0|1|2}--标识值，表明新的单位被转换为那个单位：0=meters, 1=degrees, 2=meters^2

下面给出了每一种类型的定义，这些定义都可以加入到 useradd.txt 文件中：

{units} {Feet} {0.3048} {0}

{units} {Minutes} {0.016666667} {1}

{units} {Acres} {4046.873} {2}

五、用户自定义的 RPC 读入程序

在 ENVI 主菜单中，Map—Orthorectification—Generic RPC 选项允许用户使用自定义的函数来读一个特定格式的 RPC 文件。用户自定义的 RPC 读入程序必须放置在 save_add 目录下，这样 ENVI 每次启动的时候会将其自动编译。通过加入到 useradd.txt 文件将它和 Generic RPC 选项联系起来。

RPC 读入程序在设计时，应能够在输入文件中不包含 RPC 系数时退出。它也不能显示任何用户界面。RPC 读入程序实现后，还需要在 ENVI 中注册以便 ENVI 在导入 RPC 数据时自动调用它。

RPC 读入程序必须能够找到特定影像或文件的 RPC 系数，并将它拷贝到 ENVI 的 RPC 结构中，最后将该结构返回给 ENVI。如果找不到任何 RPC 系数，该程序不应产生任何错误，而是返回-1。该值表明该函数不能读入任何 RPC 系数。

输入到 RPC 读入程序的参数既可以是影像文件 ID，也可以使文件名。文件 ID 使用 FID 关键字。文件名使用 FNAME 关键字。ENVI 可以通过 FID 关键字或是 FNAME 关键字调用 RPC 读入程序，但二者不能同时使用。

如果输入的是 FID，表示 RPC 信息来自于 ENVI 中已经打开的文件。文件 ID 可用来区别打开的影像以及与之相联系的 RPC 信息。RPC 信息将被导入到该文件中并以 RPC 系数结构返回给 ENVI。如果从该文件中没有读到任何 RPC 信息，将返回-1 给 ENVI。

如果当前 ENVI 没有任何可用的 RPC 读入程序（包括用户自定义的 RPC 读入程序）能够从文件 ID 中导入 RPC 信息，用户将会被提示选择包含 RPC 系数的文件。该文件名将传递给每一个读入程序。如果用户选择了包含 RPC 信息的文件，该文件将会以 FNAME 关键字传递给 RPC 读入程序。返回给 ENVI 的 RPC 数据必须以 ENVI_PRC_STRUCT 结构的形式存放。该结构在需要先 ENVI 中定义，它包含了执行 RPC 转换所需的系数。该结构中的各个标签描述如下：

标签名	数据类型和大小	描述
OFFSETS	Double Array[5]	用于计算 PRC 转换归一化的偏移系数，以下列的顺序：line, sample,latitude,longtitude,height
SCALES	Double Array[5]	用于计算 PRC 转换归一化的缩放系数，以下列顺序排列：line, sample,latitude,longtitude,height
LINE_NUM_COEFF	Double Array[20]	有理多项式行计算 20 多个分子参数
LINE_DEN_COEFF	Double Array[20]	有理多项式行计算 20 多个分母参数
SAMP_NUM_COEFF	Double Array[20]	有理多项式列计算 20 多个分子参数

SAMP_DEN_COEFF	Double Array[20]	有理多项式列计算 20 多个分母参数
P_OFF	Double Scalar	列偏移，图像坐标形式，从左上角 RPC 源影像到左上角影像的子集。该值总是 0，除非影像是含有 RPC 系数的大影像的子集。
L_OFF	Double Scalar	行偏移，影像坐标系统，从左上角 RPC 源影像到左上角影像的子集。该值总是 0，除非影像是含有 RPC 系数的大影像的子集。

当编写完自定义的 RPC 读入函数后，将它保存为.pro 文件或是编译为.sav 文件，并将其放置到 save_add 目录下。save_add 目录下的文件在 ENVI 启动的时候自动编译。尽管 ENVI 已经编译过了该函数，它也必须被注册。用户必须修改 useradd.txt 加入用户定义的 RPC 读入程序。为了能够使用该 RPC 函数，在文件中加入以下一行：

{rpc reader} {rpc reader name} {rpc reader function name} {}

第一对括号表示了加入到 ENVI 的是 RPC 读入程序。第二对括号标示了新的读入程序的名称。第三对括号包含了 RPC 读入程序的程序名。第四对括号保持为空。当用户修改并保存 useradd.txt 文件，用户将可以通过 ENVI 主菜单条下的 Map—Orthorectification—Generic RPC 选项来选择新的读入程序。

例子：用户定义 RPC 读入程序

下面的例子编写自定义的 RPC 读入程序从标准的 ASCII 文件中读入 RPC 系数。在本例中，包含 RPC 系数的文件名和输入的影像文件名一致并以.rpc 作为扩展名。该 RPC 读入程序使用名称和扩展名，直接从相关影像中读入 RPC 系数。本例代码如下：

```
Function ENVI_USER_RPC_READER,FID=fileID,FNAME=filename
 compile_opt strictarr
 if(N_ELEMENTS(filename) eq 0) then begin
   if(n_elements(fileID) eq 0) then return,-1
   filename=filename+'.rpc'
 endif
  rpcCoeffs={ENVI_RPC_STRUCT}
  if(~FILE_TEST(filename)) then return,-1
  openr,unit,filename,/get_lun
value=''
for index=0,4 do begin

  readf,unit,value
  rpcCoeffs.scales[index]=double(value)
endfor
```

```
  for index=0,4 do begin
    readf,unit,value
    rpcCoeffs.offsets[index]=double(value)
endfor
  for index=0,19 do begin
    readf,unit,value
    rpcCoeffs.line_num_coeff[index]=double(value)
endfor
  for index=0,19 do begin
    readf,unit,value
    rpcCoeffs.line_den_coeff[index]=double(value)
endfor
    for index=0,19 do begin
     readf,unit,value
     rpcCoeffs.samp_num_coeff[index]=double(value)
endfor
    for index=0,19 do begin
     readf,unit,value
     rpcCoeffs.samp_den_coeff[index]=double(value)
endfor
    free_lun,unit
    return,rpcCoeffs
end
```

将代码保存到文件并命名为 envi_user_rpc_reader.pro，并将该文件将放置到 save_add 目录下。

在 RPC 读入程序保存并放置到 save_add 目录下后，将下面内容加入到 useradd.txt 文件中以进行注册：

{rpc reader} {example rpc reader} {envi_user_rpc_reader} {}

在重新启动 ENVI 后，当用户选择 Map—Orthorectification—Generic RPC 时， ENVI 将会在标准的 ENVI RPC 读入程序读取失败后自动调用用户定义的 RPC 读入程序，。

六、用户自定义的移动函数

自定义移动函数能够将用户函数和 ENVI 显示组影像窗口中的移动事件或是放大窗口中的定位事件联系起来。每当鼠标在显示组影像窗口移动或是每次放大位置被改变的时候 ENVI 将会调用用户自定义的移动函数。位置信息将传给用户函数以显示与位置相关的信息。移动函数能够显示当前影像行的原始数据。

移动函数大多在文本部件中显示用户数据，非常类似于光标位置/值工具。这类移动函数首先检查文本部件是否存在，如果不存在，就创建部件。更新时，文本部件将会被关闭以获得最新的数据。显示数据可能来源于当前影像或是其它来源，也有可能当移动程序部件被创建的时候，提示输入文件。

可以在 ENVI 的配置文件 envi.cfg 文件中定义移动函数，或是在 ENVI 的主菜单上通过选择 File—Preferences 来定义移动函数。一旦定义了移动函数，它会在每次放大窗口位置变化时被调用而不会考虑影像是否显示。自定义的移动函数也会用到文件格式来严格数据的显示。当采用自定义的格式显示数据时，新的文件格式必须要加入到 ENVI 安装目录下的 menu 目录下的 filetype.txt 文件中。在显示数据前，移动函数使用 ENVI_FILE_TYPE 来检查是否为满足要求的文件格式。

移动函数也能够使用简单的 PRINT 语句，直接将结果输出到 IDL 的日志窗口中。尽管不是那么美观，但这也是开发和调试移动函数的一种非常简单方法。可以调试成功后再在移动函数中使用文本部件。

移动函数定义为过程，并且包括显示窗口号以及 X、Y 位置作为参数。X、Y 位置是浮点型数，能够表示一个像素的小数部分。同时关键字 XSTART、YSTART 定义了文件中第一像素的 X、Y 位置。移动函数通过加入 EVENT 关键字获取显示时的事件结构，用户可以使用 event.x, event.y, envent.press 等参数。

PRO user_move, DN, XLOC,YLOC, XSTART=XSTART,YSTART=YSTART

这里：

DN---Zoom 事件发生时的显示窗口号。

XLOC—影像坐标系下的当前 X 坐标，要转换为文件坐标系下的坐标必须减去 XSTART 的值。

YLOC --影像坐标系下的当前 Y 坐标，要转换为文件坐标系下的坐标必须减去 YSTART 的值。

XSTART – 文件中第一个像素的起始位置的 X 值。

YSTART --文件中第一个像素的起始位置的 Y 值。

下面的例子进一步说明了如何实现用户自定义的移动函数。

例 1：简单用户自定义的移动函数

本例构建了一个简单的移动函数，它将当前的放大窗口位置和像素值输出到 IDL 日志窗口。使用 ENVI_DISP_QUERY 和 DN 获得 FID 以及当前显示的波段。ENVI_DISP_QUERY 的 COLOR 关键字用于检测当前显示的是灰度还是 RGB 图像。对于每一个显示的波段，获取 DIMS 值并将其输入到 ENVI_GET_DATA 函数中去获取当前的像素值。像素值被打印到 IDL 日志窗口中。示例代码如下：

```
pro ud_move_1,dn,xloc,yloc,xstart=xstart,ystart=ystart
;Get the file FIDs
```

```
  envi_disp_query,dn,fid=fid,pos=pos,color=color

  if(color eq 8) then nb=3 $
     else nb=1

  ;Rrint the DN and zoom location

   print,dn,xloc+1,yloc+1

  ;Print out the current pixel for each displayed band
   for i=0,nb-1 do begin
      envi_file_query,fid[i],xstart=xstart,ystart=ystart
      dims=long([0,$
            xloc-xstart,xloc-xstart,$
            yloc-ystart,yloc-ystart])
      print,envi_get_data(fid=fid[i],pos=pos[i],dims=dims)

   endfor

end
```

通过下面的步骤执行该程序：

1、 保存该程序到文件，并将其放置在 save_add 目录下

2、 启动 ENVI

3、 在 ENVI 主菜单选择 File—Preferences 设置用户定义的移动函数为 UD_MOVE_1

4、 打开一个文件并显示一个波段

5、 在 PC 上打开 IDL 开发环境或者是 ENVI 的 shell 窗口（UNIX，MacOSX）

6、 移动窗口位置，移动函数将会将数据打印在 IDL 日志窗口。

例 2：部件式的用户自定义移动函数

本例构建了一个使用文本部件的移动函数，它将当前的放大窗口位置和像素值输出到一个文本部件中显示。程序首先检查文本部件是否存正，如果没有找到，创建一个新的部件。文本信息被输出到该文本部件。

示例代码如下：

```
Pro ud_move_2,dn,xloc,yloc,xstart=xstart,ystart=ystart
```

```
  common ud_move_2_c,ud_wid,data
    ;
;Check for a valid widget id. If the widget ID is not valid
; then Create the widget, otherwise update the text field.

if (n_elements(ud_wid) eq 0) then ud_wid=-1L
if(widget_info(ud_wid,/valid) eq 0) then begin
  ;
; Create the widget used to display the data. Give it a title and
; use envi_center to center the widget on the screen. A text widget
; is created as a place holder for the user text data.
    title='Custom Move Routine'
    envi_center,xoff,yoff
    base=widget_base(title=title,xoff=xoff,yoff=yoff,$
          /row,group=envi_main_base())
       sb=widget_base(base,/column,/frame)
    sb1=widget_base(sb,/col)
    lab=widget_label(sb1,value='Line Header Data')
    tw=widget_text(sb1,value='Data display area.',xs=40,ys=5)
    widget_control,base,/realize
       ;
    ; Use the data structure for any info the you would like to
    ; keep around.
    ;
    data={tw:tw}
    ud_wid=base

    endif

    ;
    ;Update the text widget with current information
    ;For now just display the dn,xloc,yloc,xstart,ystart.

    msg=['Display Number '+string(dn),'Loc(x,y):'+string(xloc+1)+','+$
```

```
                                        string(yloc+1),'Start(x,y):'+string(xstart+1)+','+$
                                        string(ystart+1)]
    widget_control,data.tw,set_value=msg,/no_copy

end
```

通过下面的步骤执行这个例子：

1、保存文件并将其加入到 save_add 目录下

2、启动 ENVI

3、通过 ENVI 主菜单选择 File->Preferences 设置用户定义移动函数为 UD_MOVE_2

4、打开文件并选择一个波段

5、移动放大位置，移动函数将数据输出到文本部件中。

第七章 自定义文件输入

ENVI 提供了非常强大的读写工具以导入不能直接支持的文件格式。实际上，通过交互式的设定采样数、行数、波段数、数据类型、头偏移，以及数据存储的方式，许多文件能够被 ENVI 打开。这些文件中的数据必须以 BSQ、BIL、BIP 的格式进行存储。一种更加自动的方法是将文件头信息作为参数传给 ENVI_SETUP_HEAD 以打开文件。自定义读写工具的编写仅需要解析头文件并打开文件。一旦文件被打开，ENVI 负责处理其它 I/O。

当文件不能够兼容任何标准的存储格式，用户仍然能够获取数据而不必转换数据。通过创建空间和波谱读取函数，数据能够快速读入到 ENVI 中。正如名称所示，空间读取函数处理所有的空间数据请求，从整个影像到单一像素。波谱读取函数负责所有波谱数据请求。ENVI 将所有的输入数据需求分解为这两种基本的类型，能够方便与自定义的读取函数进整合。

一、解析影像文件头

为了自动导入目前还不支持的文件格式，用户必需编写一个解析程序提取出基本的文件信息。如果数据文件不是以 BSQ、BIL 或 BIP 格式存储的。必须要编写一个自定义的读取程序。最简单来说，必须要提供下面的信息提供以便 ENVI 能够打开文件：

采样、行、以及波段的数目

数据类型

影像数据的偏移

数据的存储方式，BSQ、BIL、BIP

字节的存放顺序，Host(Intel)，Network(IEEE)

其它的文件信息也可以从文件头中解析出来，包括地理信息。该文件可以通过使用 ENVI_SETUP_HEAD 函数解析并将获取信息设为关键字值。通过使用 OPEN 关键字打开文件，相对应使用 WRITE 关键字将解析信息写入到一个 ENVI 头文件中。

解析文件头的步骤取决于文件头格式。一些文件头是基于关键字的，此时使用字符串变量读入时效果非常好。STRPOS 函数能够用于定位关键字，后面的字符串值能够转换为合适的数据。还有一种格式的文件头具有固定的位置和长度。这些文件头能够很简单的通过文件定位和单一读取每一个参数来解析。

下面的例子说明了如何解析文件头并在 ENVI 中打开文件。

例 1：解析关键字/值文件头

本例说明了如何解析一个关键字/值文件头以获取必要的文件参数。本例中的关键字/值是以空格的方式分隔的。其它的文件头可能通过“=”进行分隔。

本例中假设文件有 512 比特的文件头，并紧接存储 BSQ 格式的 Byte 数据。文件头有以下关键字：

SAMPLES value

LINES value

BANDS value

示例代码如下：

```
pro parse_header,fname


   buf=bytarr(512)
   openr,unit,fname,/get_lun
   readu,unit,buf
   free_lun,unit


   hdr=strupcase(string(buf))
   loc=strpos(hdr,'SAMPLES')+7
   ns=long(strmid(hdr,loc,strlen(hdr))
   loc=strpos(hdr,'LINES')+5
   nl=long(strmid(hdr,loc,strlen(hdr))
   loc=strpos(hdr,'BANDS')+5
   nb=long(strmid(hdr,loc,strlen(hdr))
   envi_setup_head,fname=fname,ns=ns,nl=nl,nb=nb,$
      data_type=1,interleave=0,offset=512,/open



end
```

例 2：解析位置固定的文件头

本例说明了如何解析具有定义位置参数的文件头。文件的读写位置设置为对应的比特值。参数值由文件头的数据格式决定的。典型的文件类型包括，byte、integer、long、floating point、以及 double 数据类型，以及 ASCII 码格式数据。示例读取二进制文件头，并获得采样、行、以及波段参数。

假设以 BSQ 格式存储的影像文件前加入 512 比特的文件头。文件头中有下面的信息。所有的数据值假设是 network（ＩＥＥＥ）格式存储的。

Bytes 20-23　采样的二进制值

Bytes 30-33　 行的二进制值

Bytes 40-43　波段数的二进制值

示例程序如下：

```
pro parse_header,fname

   openr,unit,fname,/get_lun
   ns=0L
   nl=0L
   nb=0L
   point_lun,unit,20L
   readu,unit,ns
   point_lun,unit,30L
   readu,unit,nl
   point_lun,unit,40L
   readu,unit,nb
   free_lun,unit

   ;Check to see if we need to swap

   if(byte(256,0) eq 0) then begin
      byteorder,ns,/lswap
      byteorder,nl,/lswap
      byteorder,ns,/lswap
```

```
    endif

    envi_setup_head,fname=fname,ns=ns,nl=nl,nb=nb,$
      data_type=1,interleave=0,offset=512,/open


end
```

二、自定义的影像读入程序

ENVI 的读入程序提供了强大的机制能够直接读入各种格式的数据，而无需转换。当文件不能转换为任何标准的存储格式时，就必需要创建自定义的读取程序。通过创建空间和波谱读取程序，ENVI 自动能动调用这些程序获取数据并为其它 ENVI 程序所使用。所有空间或波谱的数据请求由不同的读取程序完成。简单情况下，读取程序执行数据格式的转换，复杂情况下，读取程序和影像数据库进行连接，数据直接从数据库中获取。

空间读取程序负责所有空间数据的读取请求，从整个波段到单个像素。波谱读取程序处理所有的波谱数据读取请求，从单一的波谱和整行的谱带。所有的数据请求被分为这两种基本类型。当使用自定义读取程序打开文件时，设置 ENVI_SETUP_HEAD 程序的 READ_PROCEDURE 关键字以定义空间和波谱的读入程序。这样，ENVI 就会使用自定义的读取程序替换它内部的读取程序。

使用自定义读入程序打开的文件必需在 ENVI 安装目录下的 menu 菜单中的 filetype.txt 文件中定义。确定的文件类型允许文件有一个 ENVI 的头文件，但是该文件还是由 READ_PROCEDURES 关键字确定的程序打开。当数据文件具有 ENVI 头文件，并且以 ENVI 文件形式打开，头文件首先被读入。使用 PRE_FS 关键字将头文件信息传递给自定义的打开函数。自定义的打开函数解析文件头，调用 ENVI_SETUP_HEAD 包括 PRE_FS 以及常用的关键字。一个名为 OPEN_MYFILE 的自定义读取程序定义如下：

```
PRO open_myfile, FNAME, CANCEL=CANCEL, R_FID=R_FID, PRE_FS=PRE_FS
```

该程序中的参数描述如下：

FNAME：打开文件的文件名及路径

CANCEL: 当打开文件出现错误时，设置该关键字值为 1，其他情况设置为 0

R_FID: 设置该关键字为由 ENVI_SETUP_HEAD 返回的 FID

PRE_FS: 该值为传递到打开程序中的 ENVI 头文件解析信息

但是直接从 ENVI 菜单中打开一个自定义文件时，菜单事件处理程序首先调用 ENVI_PICKFILE 进行文件选择。文件名被传给上面定义的打开函数。该程序支持打开两种方式的文件（直接或是 ENVI 文件），并允许文件保持它们原始的格式，而且可以定义 ENVI 头文件参数，包括地图投影等信息。

自定义的读取程序通常需要特定的信息，这些信息不能通过 ENVI_FILE_QUERY 函数获得。这时可以为 ENVI_SETUP_HEAD 设置 INFO 关键字，并使用 ENVI_FILE_QUERY 函数的 H_INFO 关键字提取出这些信息。相关的 INFO 数据通过 HANDLE_VALUE 程序使用 H_INFO 句柄再次获得。

空间和波谱的读入程序详细描述如下：

空间读取程序：空间读取程序处理所有空间数据请求，从整个波段到单个像素。ENVI 中很多程序使用到空间数据，包括数据显示，处理函数，以及交互式函数等。读取程序不必关心数据请求的来源，它们只需满足数据需求即可。

空间数据需要确定 X 和 Y 的起始和终止位置，以及读取的波段。输入文件的单元号也被传递给读入函数。空间读取程序定义如下：

```
PRO myread_spatial, UNIT, R_DATA, FID, BAND, XS, YS, XE, YE, $
    _EXTRA=_EXTRA
```

这里：

参数	描述
UNIT	已经打开要读入文件的文件单元号
R_DATA	返回的数据值。读入程序在退出前必须定义该值。
FID	输入影像的 FID
BAND	需求数据的相应波段位置。BAND 是一个长整型的数值，从 0 到波段数减 1
XS	空间请求的 X 起始像素（文件坐标）
YS	空间请求的 Y 起始像素（文件坐标）
XE	空间请求的 X 终止像素（文件坐标）
YE	空间请求的 Y 终止像素（文件坐标）
_EXTRA	必须设置该关键字以收集对自定义读入程序无用的参数

读入程序从文件中读入合适的数据，并将结果存储在 R_DATA 参数中。打开和关闭文件是在外部进行的，它们通过 UNIT 参数传递文件逻辑单元号。

所有的读取程序必须使用_EXTRA 关键字来收集对自定义读取程序无用的关键字。在调用程序时如果用户提供了一个未列出的参数时，使用_EXTRA 能够阻止错误的发生。

下面的例子说明了如何使用自定义的空间读取程序：

例子：无符号整数的空间读入程序

本例定义一个空间读取程序用于模拟无符号的整型数据。

数据从 0 到 32767 映射于 0 到 32767，数据从 32768 到 65535 映射于-32768 到-1。本程序获取数据并将它们重新映射到-32768 到 32767 这个连续的范围。尽管显示数据值被改变了，但影像仍然正常显示。下面的波段允许表达式能够将影像转变为长整型并以正确的值显示数据。

Long(b1)+32768L

空间读取程序使用 ENVI_FILE_QUERY 函数以获取影像文件的必要信息。输出数据定义为整型的数组，因为这个读取函数只对整型函数起作用。通常，读取函数需要支持不同的数据类型并且需要根据文件的数据类型分配相应的数组。基于文件的交错方式，未转换的数据放置的输入数组，当完成数据的读入，函数检查 BYTE 交换并进行数据的转换。通过下面的方程，数据被转换为模拟的无符号整型数。

(r_data+(r_data lt 0) * 65536L)-32768L

示例代码如下：

```
pro utest_spatial,unit,r_data,fid,band,xs,ys,xe,ye,$
  _extra=_extra
    ;获取必要的文件信息
     envi_file_query,fid,ns=ns,nl=nl,nb=nb, $
     interleave=interleave,offset=offset,$
     byte_swap=byte_swap
     ;计算输出的大小并分配数组
     o_ns=xe-xs+1
   o_nl=ye-ys+1
    r_data=intarr(o_ns,o_nl,/nozero)
    ;根据文件交错方式读入数据
```

```
case interleave of
  0:begin
    a_offset=offset+2*(ns*nl*band+ys*ns)
    a=assoc(unit,intarr(ns,/nozero),a_offset)
    for i=0L,o_nl-1 do r_data[0,i]=a[xs:xe,i]
  end
    1: begin
      aout=assoc(unit,intarr(ns,/nozero),offset)
      for i=ys,ye do r_data[0,i-ys]=reform(aout[xs:xe,band+nb*i],/over)
        end
      2:begin
      aout=assoc(unit,intarr(nb,ns,/zoero),offset)
      for i=ys,ye do r_data[0,i-ys]=reform(aout[band,xs:xe,i],/over)
  end
    ; 检查 BYTE SWAP
  if(byte_swap) then byteorder,r_data
 ; 转换无符号整型到有符合整型
r_data[0,0]=(r_data+(r_data lt 0) *65536L)-32768L
end
```

波谱读取程序：波谱读入程序处理所有的波谱数据请求，从整个波谱到单个波谱单元。不同于空间数据请求，最大的波谱请求局限于一整行。波谱数据在 ENVI 中被一系列的处理函数和交互式函数所使用。

波谱请求确定所有读取的波段数，起始和终止的像素，以及行号。波谱读取程序也需要打开和关闭输入文件。通过使用 ENVI_FILE_QUERY 函数的 FID 获得输入的文件名。波谱读入程序定义如下：

```
PRO myread_spectral, FID, POS, XS=XS, XE=XE, Y=Y,SPECTRA=SPECTRA, $
 _EXTRA=_EXTRA
```

这里：

参数	描述
FID	输入影像文件的 FID
POS	所需波谱的波段位置。POS 为一个长整型的数组，值的范围由 0 到波段总数减 1。但所有的波段都要获取，POS 参数可以不定义。
XS	波谱需求的 X 起始像素（文件坐标）
XE	波谱需求的 X 终止像素（文件坐标）
Y	波谱需求的行号（文件坐标）
SPECTRA	存储需求波谱数据的变量
_EXTRA	该关键字必须定义以收集无用的关键字

读取程序从文件中读入合适的波谱数据并将结果存储在 SPECTRA 变量中。打开和关闭输入文件在波谱读取程序内部完成。所有的读取程序必须使用_EXTRA 关键字收集对程序无用的关键字。使用_EXTRA 关键字可以阻止调用程序时提供了不合适的关键字而导致的错误。如果 POS 没有定义，那么所有波段的波谱数据都被返回。下面的例子详细说明了如何定义和使用波谱读取程序：

例子：无符号整型数据波谱读取程序

本例定义了一个波谱读取程序模拟无符号整数值。数据值在 0 到 32767 映射到 0 到 32767，数据值在-32768 到-1 的映射到 32768 到 65535。

波谱读取程序使用 ENVI_FILE_QUERY 程序获取影像文件的必要信息，并打开输入文件。基于不同的文件交错方式，波谱数据被读入内存并存储在 SPECTRA 变量中。在数据获取完成后，函数检查 BYTE 交错并执行数据变换。执行下面的变换，数据被变换到无符号的整型数据：

(spectra+(spectra lt 0) *65536L) -32768L

示例代码如下：

```
pro utest_spectral, fid, pos, xs=xs, xe=xe, y=y, spectra=spectra, $
  _extra=extra
  envi_file_query, fid, fname=fname, ns=ns, nl=nl, nb=nb, offset=offset, $
    interleave=interleave, byte_swap=byte_swap, data_type=data_type
  o_nb = n_elements(pos)
  openr, unit, fname, /get_lun
  case interleave of
    0: begin
```

```
        loc = lindgen(nb) * ns * nl + y * ns + xs
        spectra = intarr(nb, /nozero)
        aout = assoc(unit, intarr(ns, /nozero), offset)
        for i=0L, o_nb-1 do spectra[0,i] = aout[loc[pos[i]]]
      end
      1: begin
        aout = assoc(unit, intarr(ns, nb, /nozero), offset)
        spectra = reform(aout[xs:xe,*,y])
        if (n_elements(pos) gt 0) then spectra = spectra[*,pos]
      end
      2: begin
        loc = y * ns + xs
        aout = assoc(unit, intarr(nb,ns, /nozero), offset)
        spectra = aout[*,xs:xe,y]
        if (n_elements(pos) gt 0) then spectra = spectra[pos,*]
      end
    endcase
    free_lun, unit
    ; 如果需要进行 BYTE swap
    if (byte_swap) then $
      byteorder, spectra
    ; 从有符号整型转变为无符号整型
    spectra[0,0] = (spectra + (spectra lt 0) * 65536L) - 32768L
    if (xs eq xe) then spectra = reform(spectra, /over)
end
```

空间和波谱读取函数都完成后，就构成了整个自定义的读取程序。将两个函数保存到文件并将它们放置到 ENVI 安装目录下的 save_add 目录下。在数据文件的 ENVI 头文件中加入下面这行使用自定义的读入函数获取数据。

Read procedures={utest_spatial,utest_spectral}

另一种方法，当调用 ENVI_SETUP_HEAD 函数时，使用 READ_PROCEDURE 关键字

使用自定义的读入程序：

Read_procedure=[‘utest_spatial’,’utest_spectral’]

执行下面的步骤执行本例：

1、 将两个读写程序保存到文件并放置到 ENVI 安装目录下的 save_add 目录下

2、 编辑 ENVI 的头文件加入下面这行内容：

Read procedures={utest_spatial,utest_spectral}

3、 启动 ENVI

4、 打开数据文件并显示一个影像

5、 使用指针位置/值工具或是交互式直方图查看修改后的数据范围。

第八章 ENVI 编程的其他主题

一、ENVI 的坐标系统

ENVI 中使用了几种不同类型的坐标系统，一些用于确定影像窗口中像素的位置，其它的表示影像数据在数组变量或文件中的位置。为了避免混淆，不论是 ENVI 编程还是通常的交互式使用 ENVI，了解这些不同坐标系统的区别是非常重要的。

1、影像坐标（像素坐标）

影像坐标值是 ENVI 显示组窗口中像素的位置，是常用的（列、行）坐标。影像坐标相当简单，它们总是随着采样数和行数增加而增加。对于列坐标，当用户在 ENVI 显示窗口中向右移动时就增加。行坐标将会从顶部到底部增加。

影像的第一个像素坐标由头文件中的 XSTART 和 YSTART 的值定义。对于大多数影像，ENVI 默认设置 XSTART 和 YSTART 的值为 1，即影像第一个像素的坐标为（1，1）。这样，如果影像是 IDL 的二维变量，下标位置为[0,0]的数据就相对应于影像坐标的(1,1)。如果 XSTART 和 YSTART 的设置为其它值，影像坐标将从该坐标值开始增加。

2、文件坐标

文件坐标是影像像素在 IDL 数组中的位置，它等于 IDL 数组的下标。不同于影像坐标，文件坐标都是基于 0 的，因为 IDL 数组的下标从 0 开始。

3、XSTART 和 YSTART

使用 XSTART 和 YSTART 值来定义影像坐标的开始值，使得 ENVI 系统中能够通用的坐标系统进行影像的参考。例如，如果小影像是由大影像的空间子集提取而来，将 XSTART 和 YSTART 的值设置为空间子集的起始列和行的位置，就能够保证空间裁减的影像依然保持它原来的坐标值。在多数情况下，ENVI 处理程序自动为结果影像设定合适的 XSTART 和 YSTART 值。例如：在执行影像对影像的配准时，被配准的影像的 XSTART 和 YSTART 值和参考影像的坐标相关。这就允许这两幅影像使用一个共同的影像坐标系统进行直接比较。（用户可以通过对配准结果和参考影像进行动态链接看出效果，链接的偏移是直接由 XSTART 和 YSTART 值计算获得的。）

4、XSTART 和 YSTART 的编程

如果用户用过 ENIV 主菜单中的 File->Edit ENVI Header 工具编辑影像的 X、YSTART 的值，会发现文件坐标和影像坐标的关系如下：

(sample)影像坐标＝文件坐标+XSTART

（Line）影像坐标＝文件坐标＋YSTART

然而，必须要注意的是，交互式 ENVI 中获得 XSTART 和 YSTART 的值和在 ENVI 批处理模式下获得值是不同的。在批处理模式下，XSTART 和 YSTART 的值由 ENVI_FILE_QUERY 函数返回，并且是从 0 起始的值，它们总是比 ENVI 文件头信息对话框显示的值少 1。例如：如果一个文件具有标准的 XSTART 和 YSTART 值为 1，在 ENVI 的主菜单条选择 File->Edit ENVI Header 选项显示值也为 1。但是在批处理中使用同样的文件，ENVI_FILE_QUERY 的返回 XSTART 和 YSTART 的值为 0。因此，在批处理模式下，影像坐标和文件坐标的关系如下：

（sample）影像坐标＝文件坐标＋XSTART＋1

（line）影像坐标＝文件坐标＋YSTART＋1

在批处理模式下使用 ENVI_SETUP_HEAD 或 ENVI_ENTER_DATA 定义一个新的文件时，XSTART 和 YSTART 都将定义为基于 0 的数值。要获得一个第一个像素的影像坐标为（1,1）的影像，设置 XSTART 和 YSTART 的值为 0。

在编写用户函数时要处理并返回影像数据，如果允许用户能够对输入的影像进行空间裁决，为结果输出影像最终正确的 XSTART 和 YSTART 的值是非常重要的。(这样结果影像就能够和原始影像坐标相匹配)。下面的部分代码说明一个更新坐标的方法：

```
; select an input image and return the DIMS
ENVI_SELECT, fid=fid,dims=dims ...
;get the image's XSTART and YSTART values
ENVI_FILE_QUERY, fid,xstart=xs,ystart=ys ...
;if the starting smaple and line are not zero
; then it was spatially subsetted so you'll
;need to update the XSTART and YSTART for the
; output image's header file
```

IF (dims[1] ne 0) then xs=xs+dims[1]

IF (dims[3] ne 0) then ys=ys+dims[3]

由于批处理模式下，XSTART 和 YSTART 的值是基于 0 开始的数，可以将它们直接加到 DIMS 值上。当写入处理文件的头文件时，XSTART 和 YSTART 的值由 XS 和 YS 的变量值设置。

二、感兴趣区（ROI）

感兴趣区（ROIs）通常是用户在影像上选择绘出的空间子区。这些区域通常是不规则形状的，主要用于为分类、掩模或是其它操作提供统计信息。ENVI 允许选择任意组合的多边形、矢量或点作为 ROI。一幅影像可以定义多个 ROI，并可以为其它的影像所使用。

1、感兴趣区处理

在图形上来说，一个感兴趣区可以是一系列的多边形、线形或是多个点组成的。但从处理的角度来看，ROI 就是和数据关联的地址。大多数的 ROI 处理程序并不需要图形上的关系，而是需要和 ROI 相连系的数据。例如：一个 ROI 内的平均波谱是计算每个波段的所有像素值的和并除去 ROI 内的共像素点数，而不用考虑哪些点位于 ROI 内。用户也只需要知道用于计算均值的每个 ROI 点的地址。

ENVI 提供了一系列函数进行 ROI 的处理。这些函数能够打开 ROI 文件，列出所有打开的 ROI 文件信息，获取 ROI 地址，取得数据，将 ROI ids 转换为 DIMS 中的指针，以及创建新的 ROI。在处理 ROI 的时候，ROI 和数据文件必需要有一个关联。在 ENVI 中 ROI 通过文件大小和文件相关联。通过选择文件并获取和该文件空间大小相匹配的 ROI 列表。所选择的文件并不需要是绘制 ROI 的文件，它只需和原始文件具有相同的空间大小。接着，需要的 ROI 被选择并使用文件的相关信息以获得对应的数据。整个 ROI 数据以一个单一数组的形式返回（在 ROI 处理中没有分块处理）。

注：ROI 和通过空间纬度，采样数、行数和文件关联的。使用 ROI 协调处理程序可以将 ROI 映射到一个不同空间纬度的文件上。

下表列出了 ENVI 提供的 ROI 函数：

函数名称	描述
ENVI_CREATE_ROI	创建一个新的 ROI
ENVI_DEFINE_ROI	向一个 ROI 中添加对象
ENVI_DELETE_ROIS	从 ENVI 中删除 ROIs
ENVI_GET_ROI	获得一个 ROI 的地址
ENVI_GET_ROI_DATA	获得 ROI 向对应的数据
ENVI_GET_ROI_DIMS_PTR	将 ROI id 转换为 DIMS 指针值
ENVI_GET_ROI_IDS	获取 ROI ids 列表
ENVI_RESTORE_ROIS	打开并导入一个保存的 ROI 文件
ENIV_SAVE_ROIS	保存 ENVI 中的 ROI

2、选取 ROI

为了理解 ROI 的选择， ROI 是和一定的采样数和行数相关联的。对于一个给定的空间大小，可以具有任意数目的关联 ROI。ENVI_GET_ROI_IDS 返回了与给定的空间大小相关联的所有可用 ROI。ENVI_GET_ROI_IDS 可以通过三种方式确定需要的空间大小：

使用关键字 NS 和 NL

使用 FID 关键字，和由 FID 确定的文件大小匹配

使用 DN 关键字，和由 DN 确定的影像大小匹配

复合部件 WIDGET_MULTI 为 ROI 的选取提供了非常好的方法。一旦最终 ROI 被选择，就可以通过它们来获取相关联的 ROI 数据，并输入到处理程序如统计，或是获取 ROI 地址。

除了当前 ENVI 进程中定义的 ROI，任何先前保存的 ROI 文件也可以通过 ENVI_RESTORE_ROIS 读入到内存中。文件中的所有 ROI 都被读入，ENVI_RESTORE_ROIS 不返回任何 ROI 的 ids，可以使用 ENVI_GET_ROI_IDS 获取导入的 ROIs。下面的例子说明了如何进行 ROI 选择：

例子：ROI 选择

在本例中，通过 ENVI_GET_ROI_IDS 及一个相关联的空间大小来选取基本的 ROI。首先在一幅影像上交互式的绘制三个 ROI。影像文件通过 ENVI_SELECT 选取，空间大小通过 ENVI_FILE_QUERY 获取。接着通过确定文件的 NS 和 NL 值获取 ROI ids。为了比较，ROI ids 也通过影像的 FID 来获取。下列步骤列出了整个处理过程：

1、 启动 ENVI，打开一个文件，并显示一个灰度影像；

2、 交互式的绘制三个分开的多边形 ROI，每个大于 200 个点；

3、 在 PC 上打开 IDL 的开发环境，或是 ENVI 的 shell 窗口（UNIX、Mac OS X）；

4、 在 ENVI 的命令行向输入以下命令；

5、 选取文件：

Envi_select,title=’Input Filename’,fid=fid

6、 获取采样数和行数：

Envi_file_query,fid,ns=ns,nl=nl

7、 获取相关联的 ROIs，并打印结果

Roi_ids=envi_get_roi_ids(ns=ns,nl=nl)

Print,roi_ids

8、 由 FID 获取 ROIs

Roi_ids=envi_get_roi_ids(fid=fid)

Print,roi_ids

两次打印处理的 ROI_IDS 的值应该一致，因为它们都是引用了同样 ROI。

例子：ROI 选取和 WIDGET_MULTI

本例扩展了前面的例子，使用 WIDGET_MULTI 来进行 ROI 的选取。

首先选择一个文件用于为 ROI 选取提供空间大小。ENVI_GET_ROI_IDS 返回了 ROI 的 ids，并通过 ROI_NAMES 关键字返回的相关连的 ROI 名称。如果没有找到相关联的 ROI，将返回－1 值，程序打印错误信息并退出。创建一个自动事件管理的小部件用于进行最终的 ROI 选取。列表中选择的项目所标识的变量的 PTR 以及 ROI 名称以及 ids 被打印出来。

示例代码如下：

```
pro roi_multi_sel
  envi_select,title='Input Filename',fid=fid
  if(fid eq -1) then return
    roi_ids=envi_get_roi_ids(fid=fid,$
      roi_name=roi_names)
  if(roi_ids[0] eq -1) then begin
    print,'No regions associated with the selected file'
    return
  endif
   ; Compound widget for ROI selection
 base=widget_auto_base(title='ROI Selection')
```

```
wm=widget_multi(base,list=roi_names,uvalue='list',/auto)
result=auto_wid_mng(base)
if(result.accept eq 0) then return
ptr=where(result.list eq 1)
print,roi_names[ptr]
print,roi_ids[ptr]
end
```

通过下面的步骤运行本例：

1、 保存该程序到文件并将其放置在 save_add 目录下

2、 启动或从新启动 ENVI 并打开一个文件，显示灰度影像

3、 交互式的绘制三个分开的多边形 ROI，每一个在 200 个点左右。

4、 转到 IDL 开发环境或是 ENVI 的 shell 窗口

5、 在 ENVI 命令行模式下输入 roi_multi_sel

例子：恢复保存的 ROIs

本例交互式的恢复一个保存的 ROI 文件。使用复合部件 ENVI_PICKFILE 用来选择文件并返回文件名称，文件名称传递给 ENVI_RESTORE_ROIS。一个信息对话框显示恢复的 ROIs 信息。

1、 启动 ENVI

2、 打开和显示一个灰度影像

3、 交互式的绘制三个分开的多边形 ROIs，每一个有 200 点左右

4、 保存 ROI 到一个文件

5、 打开 IDL 开发环境或是 ENVI 的 shell 窗口

6、 在 ENVI 命令行下输入：

```
Name=envi_pickfile(title='ROI File',filter='*.roi')
```

7、 输入下面的命令恢复保存的 ROI

```
Envi_restore_rois,name
```

现在 ENVI 中就有两组 ROI，一组是交互式创建的，一组是从文件中读取的。尽管两组来源都相同，但现在它们被认为是有区别的。

3、使用 ROI 数据

一旦选取了 ROI，获取和处理数据就非常简单。ROI 数据通过使用 ENVI_GET_ROI_DATA 获得。获取数据的文件由 FID 关键字传给 ENVI_GET_ROI_DATA 函数。

下面的例子在先前 ROI 选择的基础上加入了数据获取部分。

例子：使用 ROI 数据

使用 ENVI_GET_ROI_IDS 进行相关空间纬度基本的 ROI 选取。本例中交互式选取和由 FID 确定的空间大小相关联的所有 ROI。ROI 区域内第一波段内的数据使用 ENVI_GET_ROI_DATA 获取。该波段内 ROI 均值进行计算并输出。

1、 启动 ENVI，打开文件，显示灰度影像。

2、 交互式绘制一个多边形 ROI，大约在 200 像素点。

3、 打开 IDL 开发环境（PC）或 ENVI shell 窗口（UNIX、Mac OS X）.

4、 在 ENVI 的命令行输入下面的命令：

```
Envi_select, title=’Input Filename’, fid=fid, pos=pos
```

5、 输入下面命令获取由 FID 确定的空间大小的 ROI

```
Roi_ids=envi_get_roi_ids(fid=fid)
```

6、 输入下列命令获取第一波段的 ROI 数据，并计算 ROI 均值

```
Data=envi_get_roi_data(roi_ids[0],fid=fid,pos=pos[0])
Print, ‘ROI mean =’,total(data)/n_elements(data)
```

要获取 POS 数组中其它波段的数据，再次调用 ENVI_GET_ROI_DATA 程序，将 POS 关键字设置为相应的波段。该处理过程需要进行循环，直到所有的波段数据都被获取。或者所有波段的 ROI 数据都能够简单的调用 ENVI_GET_ROI_DATA 获取。这个交互式的例子仅仅是做为说明，在实际处理中它将做为用户函数的一部分。

例子：计算 ROI 均值

本例扩展了先前的使用 WIDGET_MULTI 选取 ROI 区域的程序。它获取 ROI 数据并计算 ROI 均值。

本例使用 ENVI_SELECT 选择用于空间大小参考的文件。ENVI_GET_ROI_IDS 用于返回 ROI ids 列表，其中关键字 ROI_NAMES 返回相关联的 ROI 名称，如果没有找到 ROIs，返

回单个值为－1 的数组，程序打印错误信息并退出。下一步，创建一个自动事件管理的小部件进行最终的 ROI 选择。列表中被选择的项目由 PTR 变量标识。选定 ROI 区域内每个波段的数据被读入并计算均值，打印出结果。

示例代码如下：

```
pro roi_mean
 envi_select,title='Input Filename',fid=fid,pos=pos
 if(fid eq -1) then return
    roi_ids=envi_get_roi_ids(fid=fid,roi_names=roi_names)
 if(roid_ids[0] eq -1) then begin
      print,'No regions associated with the selected file'
      return
 endif
  ;Compound widget for ROI selection
 base=widget_auto_base(title='ROI Selection')
 wm=widget_mulit(base,list=roi_names,uvalue='list',/auto)
 result=auto_wid_mng(base)
 if(result.accept eq 0) then return
 ptr=where(result.list eq 1,count)
 iresult=dblarr(n_elements(pos))
 ;ROI Mean calculation
 for i=0L,count-1 do begin
      for j=0L,n_elements(pos)-1 do begin
         data=envi_get_roi_data(roi_ids[ptr[i]],fid=fid,pos=pos[j])
         result[j]=total(data,/double)/n_elements(data)
      endfor
      print,roi_names[ptr[i]]
      print,result
 endfor
 end
```

通过下面的步骤执行本例：

（1）保存程序到文件并将其放置在 save_add 目录下

（2）启动或重新启动 ENVI

（3）打开一个文件并显示灰度影像

（4）交互式的绘制三个分开的多边形 ROI，每一个大约 200 个点

（5）打开 IDL 开发环境或是 ENVI 的 shell 窗口，输入下面的命令：roi_mean

4、使用 ROI DIMS 指针

ENVI 中的一些函数需要使用 ROI 来获得空间子集。在这种情况下，DIMS 数组的第一元素为指向特定 ROI 的指针。ENVI 函数 ENVI_GET_ROI_DIMS_PTR 将 ROI 的 ids 转换为一个 DIMS 指针。

注意：当 ROI 区增加或减少的时候，ROI DIMS 指针值会发生改变，因此最好的方法是在定义 DIMS 数组的时候进行 ROI ids 的转换。

例子：ROI DIMS 指针

本例将上面计算 ROI 均值的例子进行修改，使用 ENVI_STATS_DOIT 计算 ROI 的统计值。

示例代码如下：

```
pro roi_stat
  envi_select,title='Input Filename',fid=fid,pos=pos
  if(fid eq -1) then return
  roi_ids=envi_get_roi_ids(fid=fid,roi_names=roi_names)
  if(roi_ids[0] eq -1) then begin
    print,'没有区域和选择文件相匹配'
    return
  endif

  ;ROI 选取的复合部件
  base=widget_auto_base(title='ROI Selection')
  wm=widget_multi(base,list=roi_names,uvalue='list',/auto)
  result=auto_wid_mng(base)
  if(result.accept eq 0) then return
```

```
  ptr=where(result.list eq 1,count)
  result=dblarr(n_elements(pos))
  ;ROI 统计计算
  for i=0L,count-1 do begin
      dims=[envi_get_roi_dims_ptr(roi_ids[ptr[i]]),0,0,0,0]
      envi_stats_doit,fid=fid,dims=dims,pos=pos,comp_flag=0,$
      report_flag=0,mean=mean,stdv=stdv,dmin=dmin,dmax=dmax
      print,roi_names[ptr[i]]
      print,dmin,dmax,mean,stdv
  endfor
end
```

通过下面的步骤执行本例：

（1）保存程序到文件并将其放置在 save_add 目录下

（2）启动或重新启动 ENVI

（3）打开一个文件并显示灰度影像

（4）交互式的绘制三个分开的多边形 ROI，每一个大约 200 个点

（5）打开 IDL 开发环境或是 ENVI 的 shell 窗口，输入下面的命令：roi_stat

5、使用 ROI 地址

ENVI 通过使用 ROI 的地址提供了访问任何 ROI 空间位置的方法。ROI 的地址是一个单个元素的地址。该地址以影像的第一个像素为参考并在采样的方向增加。下表列出了一些 ROI 地址以及和它们相对应的显示位置。影像大小为 10samples*8lines

ROI Address	Corresponding pixel
0	影像的第一个像素
12	第二行的第三个像素
30	第三行的第一像素
79	影像的最后一个像素

ENVI_GET_ROI 函数返回了特定 ROI 的地址。如果需要的话，地址能够转换为空间的 X 和 Y 值。ROI 的地址是基于 0 的，影像的第一个像素的地址为 0。

例子：ROI 地址

本例计算一个 ROI 地址的 X 和 Y 值。

通过下面的步骤来执行本程序：

（1） 启动 ENVI

（2） 打开一个文件并显示灰度影像

（3） 交互式绘制一个多边形的 ROI，大约 200 个点

（4） 打开 IDL 开发环境或是 ENVI 的 shell 窗口

（5） 在 ENVI 命令行下输入：

Envi_select, title='Input Filename', fid=fid

（6） 输入下面的命令：

Roi_ids=envi_get_roi_ids(fid=fid)

（7） 获取 ROI 地址：addr=envi_get_roi(roi_ids[0])

（8） 获取文件的行数和列数：envi_file_query,fid,ns=ns, nl=nl

（9） 计算并打印 ROI 中每个点的（X、Y）值

```
Y=addr/ns
X=addr-y*ns
Print,x
Print,y
```