IDA — A Tcl Imaging and Data Analysis extension

Version 0.5 12 April 1995

Paul Alexander, Cavendish Laboratory, Cambridge, UK

1 Introduction

IDA is an image/data processing and analysis language based on Tcl (Ousterhout 1994). The language is implemented as a Tcl extension and may be used with other extensions. It provides the tools necessary for image/data analysis, but not for display although *hooks* are provided within the Tcl **pgplot** extension to provide display functions.

This version provides support for *images* alone — in this context we consider an image to be a set of gridded data values represented by a contnuous 4D volume. Each element in this volume is referred to as a pixel (for 3D data the term voxel is often used). At each pixel we may have a vector of data values — therefore 5-dimensional data may be handled. A normal representation would be to map the 4 dimensions of the data volume onto three spatial and one time dimension and to hold in the vector any number of *physical* quantities. These may be for example density, pressure, temperature in a fluid-dynamical simulation, red, green and blue for a true-colour image or components of one or more complex quantities. Future versions will include support for a second data format where the data consist of a vector of values on discrete points in an N-dimensional space and may include connectivities between the data — such a format is more appropriate for representing say the output of N-body simulations or molecular data.

The internal representation is as a 5-dimension array of 4-byte floating point values; this is a necessary representation for the — **IDA** can also be used to analyse images more usually represented with less precision although at a cost in terms of memory usage and for some operations processing time (but not all). Fixing on one data type greatly simplifies the implementation and is aimed at providing functionality rather than the greatest efficiency. [1]

The remainder of this section introduces a number of terms and concepts used throughout the package.

1.1 Image identifiers – imId

All commands acting on images take an image identifier — in the documentation this is abreviated to **imId**. Commands will also return an **imId** if they modify an image. In the current implementation of the package **imId**s are simply an integer value, in a future release this will change to be an integer prefixed by a short string (probably *img*) to make the behaviour consistent with other Tcl extensions. You should not therefore *rely* on **imId**s being integers.

[1]It is the author's view that the power of modern workstations is so great that many issues which were until recently of concern — memory usage, floating-point performance etc. — are no-longer overiding considerations in the design of data-processing software. Of much greater improtance is the time required for scientists to implement relatively efficient, and accurate code.

1.2 Data formats

At present three image formats are supported and identified via a numerical *type*:

(1) MRAO map format.

(2) DDF format — this stands for *Directory Data Format* and is a very flexible format supported by the **IDA** extension. It is the prefered format for **IDA** images.

(3) FITS — Flexible image transport.

Note that in the current release of the **IDA** little attention is paid to image headers beyond those items needed to read and write data. You should not therefore use the current implemntation to convert between image formats and only DDF format has been tested to any extent. In future versions of the package there will be proper support for image headers and for image conversion.

1.3 Sub images

Various commands take as some of their optional arguments a subimage. A sub-image specifies a contiguous region of the image, for example a range of values from a 1D image or a rectangular region in a 2D image. To specify the subimage you must indicate the bouding pixel in each dimension; a standard notation, common to all imaging commands, exists to specify the subimage by specify options in the standard Tcl manner:

```
-x1 Ix1 -x2 Ix2
-y1 Iy1 -y2 Iy2
-z1 Iz1 -z2 Iz2
-t1 It1 -t2 It2
-v1 Iv1 -v2 Iv2
```

throughout this text the term **subimage** will refer to a region of the image specified using the above notation for a sub-image. Any optional parameters which are not supplied default to the full extent of the image. It is legal to specify a subimage which extends beyond the boundary of the image — only those pixels which are within the image will be used in processing.

1.4 Blanking

All imaging commands support the concept of a *blank* pixel. This is a pixel which has been set to a particular value to indicate that it does not contain a physically realistic number. The value is usually chosen to be completely out of the bounds of physically sensible values (usually a very large negative value such as 1.0E-30). Commands exist to blank and un-blank regions of images. The blanking value is usually set in the image header or definition record for an image stored on disc.

1.5 Normalization

Imaging commands which sum the image make use of a normalization parameter. The normalization parameter is usually defined in the image header or definition record for an image stored on disc.

2 Commands to perform basic functions on images

2.1 The img_image command

The **img_image** command is used to provide low-level access to images and to perform image I/O. The synopsis for the command is:

```
img_image option ?parameters?
```

Table 1 provides a summary of the available options. The most important use for the **img_image** command is to perform I/O for images. An example illustrates the use of the **img_image** command to read in data and display it using the **pgplot** extension which defines a new command **pg**.

```
set n [img_image read -file ex1.dat -type 1]
pg stream open 1 /xwindow
set st [img_anal $n statistics]
pg image $n [lindex $st 0] [lindex $st 1]
```

In this example the **img_anal** command (Section 2.2) is used to annalyse the image and obtain statistics which are needed for **pg image** — this command takes arguments which are the minimum and maximum of the range to display.

2.2 Image analysis with img_anal

Analysis of images is performed using the **img_anal** command. The synopsis for the command is:

```
img_anal imId option ?parameters? ?subimage?
```

Options for the command are listed in Table 2. For all forms of the command you may specify a **subimage** (see section 1.2) which should appear after all compulsary arguments; only pixels within the specified subimage are analysed. The following example illustrates how the output of the **statistics** option can be used to implement an algorithm which attempts an analysis of noise in an image by assuming highly discrepant points are in fact true signal.

Options	**Notes**
read ?-file **file**? ?-type **type**?	Read a data file from **file**. The format of the data is specified by the parameter **type** and has one of the values listed in Section 1.2.
write ?-imid **I**? ?-file **file**? ?-type **type**?	Read a data file from **file**. The format of the data is specified by the parameter **type** and has one of the values listed in Section 1.2.
destroy **imId**	destroy the data structure associated wtih **imId**
create **defn**	create a new image with dimensions given by **defn** which is a normal Tcl array which should have all the elements required for a definition record. One method of ensuring this would be to read the definition record from another **imId** and modify it.
collapse **imId**	reduce dimansionality of **imId** by collapsing any dimension of unity. This option sets range on dimensions of unity to 1.
next	return next free image ID **imId**
exists **imId**	check whether **imId** is valid and exists. The command returns a boolean indicating true (1) if **imId** exists.
max	return maximum number of images that may be allocated.
defget **imId defn**	get the definition record, returned in array **defn**
defset **imId defn**	set the definition record for **imId** from the array **defn**.
defread **file defn**	read the definition record for file **file**; the definition record is returned in array **defn**.
defputs **imId**	display the definition record for **imId** on stdout

Table 1: Options and return values for the **img_image** command.

Options	**Notes**
statistics ?-gate *g*? ?-agate *ag*? ?-min *m*1? ?-max *m*2?	obtain statistics for an image. Only pixels satisfying: $$val > g \ \&\& \ abs(val) > ag \ \&\& \ m1 \leq val \leq m2$$ are analysed in the specified subimage. A list of {max min mean sd min-pixel max-pixel} is returned. The defaults are such as to analyse all pixels in the image.
maxmin ?-gate *g*? ?-agate *ag*? ?-min *m*1? ?-max *m*2?	obtain the maximum and minimum for an image. Only pixels satisfying: $$val > g \ \&\& \ abs(val) > ag \ \&\& \ m1 \leq val \leq m2$$ are analysed in the specified subimage. A list of {max min min-pixel max-pixel} is returned. The defaults are such as to analyse all pixels in the image.
sum ?-gate *g*? ?-agate *ag*? ?-min *m*1? ?-max *m*2?	obtain the sum of pixel intensities for an image. Only pixels satisfying: $$val > g \ \&\& \ abs(val) > ag \ \&\& \ m1 \leq val \leq m2$$ are summed. The sum of pixel intensities satisfying the above constraints and within the specified subimage is returned scaled by the normalization parameter. The defaults are such as to analyse all pixels in the image.
histogram ?-ndata **ndata**? ?-low **low**? ?-high **high**?	Extract a 1D image which is a histogram of image data in the specified sub-image of the image. The command returns the imId of the new image which has xdim = **ndata** and vdim=2; the first vector argument is the value for the bin, the second is the number of pixels within the bin. This format is appropriate for display using the **pg** extension. The default for **ndata** is 100 and **low/high** default to the min/max on the image.

Table 2: The **img_anal** command for basic image analysis. The first argument to **img_anal** must be a valid **imId** and the second one of the above options. All the options take as an additional argument a **subimage** specification to limit the analysis to a specified sub-region of the image. The **subimage** specification should occur after all compulsary arrguments.

```
proc img_noise { imId nSD} {
 # determine mean and SD of image data
   set st [img_anal $imId statistics]
 # discard points more than nSD standard deviations from the mean
 # and recalculate the statistcis for the image.
 # Iterate three times
   for {set n 0} {$n < 3} {incr n} {
      set min [expr [lindex $st 2] - $nSD * [lindex $st 3]]
      set max [expr [lindex $st 2] + $nSD * [lindex $st 3]]
      set st [img_anal $imId statistics -min $min -max $max]
   }
   return [lindex $st 3]
}
```

2.3 Modifying images with img_apply

The command **img_apply** takes an image and modifies it according to a series of options which are applied sequentially to the image. The first agument to the command must be an **imId** and subsequent arguments are modifier options which themselves may take additional arguments as either parameter values of other **imId**s of other images. The range of options available with the **img_apply** command is given in Tables 3 and 4.

```
img_apply imId ?subimage? ?option ?parameter?? ?option ?parameter?? ...
```

A **subimage** may be specified which limits the command to operate on the specified subimage — the subimage may be specified anywhere on the command line provided it does not break up an option which has to be followed by a specified set of parameters; for clarity it is recommended that the sub-image specification follows directly after **imId**.

For example the following command averages four images:

```
img_apply 0 += 1 += 2 += 3 scale 0.25
```

In the following example we quantizes an image into a set of integer levels and then displays each level in turn. To do this a temporary copy of the image is taken so that this can be used as work space:

```
# quantize the image --- note the variable img1 just contains the the imId 0
set img1 [img_apply 0 quantize 0.0 100.0 1.0]

# make a copy for use as workspace
set img2 [img_geom $img1 subimage]

# loop and display all level using the img_display procedure
```

Options	**Notes**
= **imId1**	set **imId** equal to **imId1**; the operation is applied only to the **subimage** common to both images.
== **imId1**	set **imId** equal to **imId1**; the operation is applied to the **subimage** of **imId**. Values are taken sequentially from **imId1** such that the first pixel in the subimage of **imId** is set equal to the first value of **imId1**.
+=, -=, *=, /=, **imId1**	As for "=" except: **imId** = **imId op imId1**, where **op** is +, -, *, / for addition subtraction multiplication and division respectivley.
+==, -==, *==, /==, imId1	As for "==" except: **imId** == **imId op imId1**, where **op** is +, -, *, / for addition subtraction multiplication and division respectivley.
atan2 **imId1 imId2**	set **imId** = atan2(**imId1**, **imId2**)

Table 3: Options for the img_apply command which combine images. The first argument to img_apply must be a valid **imId** and subsequent arguments are options as specified in this and the next table applied sequentially.

Options	**Notes**
log, ln, log10, exp, sqrt sin, cos, tan, asin, acos, atan	Apply specified function to imId: set **imId** = func(**imId**)
10	set **imId** = 10****imId**
raise **x**	raise **x** to power **imId**; set **imId** = **x******imId**
pow **x**	raise **imId** to power **x**; set **imId** = **imId******x**
scale **x**	scale **imId** by **x**; set **imId** = **x** * **imId**
offset **x**	offset **imId** by **x**; set **imId** = **imId** + **x**
binary **x**	force **imId** to be a binary image: if **imId** < **x** then set **imId** = 0.0, else set **imId** = 1.0
set **x**	set **imId** to the value **x**
pass **x1 x2 xI**	pass only values in interval **x1** to **x2**: if **imId** < **x1** or **imId** > **x2** then set **imId** = **xI**
range **x1 x2 op xI**	For values in the interval **x1** to **x2** apply the specified operation. **op** should be one of =, +, −, * or /. Then: if **x1** < **imId** < **x2** then set **imId** = **imId op xI**
quantize **x1 x2 xI**	quantize image to values are multiples of **xI** in the range **x1** to **x2**
blank	set image to blank values
unblank **x**	set all blank pixels to value **x**
inv	invert the image; **imId** = 1.0/**imId**

Table 4: The img_apply command for modifying images. The first argument to img_apply must be a valid **imId** and subsequent arguments are options as specified in this table applied sequentially.

```
# note how img2 is constantly reset to img1
for {set n 0} {$n < 100} {incr n} {
   set m [expr $n + 1]
   img_apply $img2 = $img1 pass ${n}.0 ${m}.5
   img_display $img2
}
img_image destroy $img2
```

This loop in this example could have been written very compactly as:

```
for {set n 0; set m 1} {$n < 100} {incr n; incr m} {
   img_display [img_apply $img2 = $img1 pass ${n}.0 ${m}.5]
}
```

Note that at the end of the operation we destroy the temporary image **img2** — in a long program without carefully destroying intermediate images the memory usage will rise quickly and you will eventually use up all the available image descriptors.

2.4 Geometrical transformations with img_geom

The **img_geom** command is used to perform geometrical transformations on images. Some of the operations result in new images others are performed "in place" — if the operation is stricly reversible then the latter method is usually followed.

Table 5 lists the options to the **img_geom** command — the first argument to **img_geom** is the **imId** of the image to transform the second is the tranformation option as listed in Table 5.

Options	**Notes**
subimage **subimage**	Extract a subimage for **imId**. The format of a subimage is described in section 1.3. A new image is created by this command and the **imId** of the new image is returned.
project axis	Project the image along the specified axis. The **axis** argument should be one of **x, y, z, t** or **v**. This reduces the dimensionality of the image by one. The projection is accomplished by summing all pixels to be projected onto a given direction. If any data selection is required (gating of the data for example) this should have already been performed.
bshift ?-dx **x**? ?-dy **y**? ?-dz **z**? ?-dt **t**?	Perform a barrel shift on the image with pixel offsets given x, y, z, t. The defaults for the pixel offsets are 0 in all directions. A barrel shift shifts the image by an integer number of pixels in all dimensions and is performed using "periodic boundary conditions" so that pixels projected outside the image will be wrapped back onto the opposite side of the image.
flip axis	Flip the image in the direction specified by axis. The **axis** argument should be one of **x, y, z, t**.
transpose axis-list	Transpose the axes of the image. The new order of the axes should be listed after the transpose option. For example the following will transpose the x and y axes for an X-dimensional image: `img_geom 0 transpose y x`
slice **subimage** ?-ndata **ndata**?	Extract a slice from the image. The subimage format is used to specify the extent of the slice which is taken from (x1, y1, z1, t1, v1) to (x2, y2, z2, t2, v2). The data are sampled along this slice to give **ndata** points. The slice is returned as a 1D image with vdim set to 2; the first vector argument is the length along the slice, the second is the interpolated image value. This format is appropriate for display using the **pg** extension. **ndata** defaults to 100.

Table 5: The img_geom command for performing geometrical transformations to images. The first argument to the command is the **imId** of the image to transform, the second is the option as listed in this table.

Options	**Notes**
transform ?-matrix {**matrix-elements**}? ?-vector {**vector-elements**}? ?-cvector {**cvector-elements**}? ?-m11 **m11**? ... ?-m44 **m44**?	Apply a generalised linear transform to the image. The transform is specified in terms of a matrix and vector whose elements may be real, floating-point, values. The transformation is applied to determine the mapping of the pixels of the new output image (the **imId** of which is returned by this command) to those of the original image. The number of elements supplied for the matrix must correspond to a square matrix and should form a propoer Tcl list. The corresponding matrix is taken as the upper left sub-matrix of the full 4D tranformation matrix. If the matrix is M, the vector V and the c-vector C, then pixels in the output image po to the input image pi are related by: $$po = V\ + M.(pi\ -\ C)\ + C$$ therefore V provides a linear translation and M a general tranformation about a point C.

Table 6: The img_geom command for performing generalised transformations to images. The first argument to the command is the **imId** of the image to transform, the second is the option as listed in this table.