TRDIDL: IMAGE DISPLAY, ANNOTATION, AND EXAMINATION WITH TIM ROBISHAW'S `display.pro` AND RELATED PROCEDURES
August 30, 2011

Carl Heiles, Tim Robishaw

Contents

1. INTRODUCTION

Tim Robishaw's `display.pro` does most of what we cover in other handouts[1], but much more simply. Here's an introduction to this and related programs. We emphasize that this is

[1]Particularly the three handouts "IDIDL—Image Display and Manipulation in IDL" (ididl.ps) "1d2d3d: One, Two, and Three Dimensional Color Images" (newdimensionalcolor.ps). "IACIDL: Image Annotation and Colorbars in IDL" (annotated_images.ps).

an *introduction* to those programs; for more details, look at their documentation (in IDL, that's `doc_library, 'display'` (or, if you use our startupfile, `doc, 'display'`) for `display.pro`).

Important! There are various authors who wrote different versions of `display` and `colorbar`; we treat, and recommend, *Robishaw's* versions. The Robishaw procedures we cover here include:

1. `display.pro` . This displays 1d or 3d images, draws axes, labels axes, properly bytescales the images (either automatically or according to user specs), changes contrast...
2. `trc.pro` . Shorthand for trcursor, this slaps a nice crosshairs on the screen and, when you push the mouse button, records the image value and the x/y positions. `trc` works for ordinary graphs, too, for which it is very handy!
3. `trp.pro` . Shorthand for trprofiles, this plots image profiles (the image intensity versus the x or y coordinate).
4. `colorbar.pro` . Puts a colorbar near your displayed image.

In short, the first three of these Robishaw procedures do what the native IDL procedures `tv`, `rdpix`, and `profiles` do, except that they can give the x/y values in designated coordinate values instead of pixels, do a few other nice things, and work better.

2. USING TIM ROBISHAW'S `display` TO MAKE AND EXAMINE 1D OR 3D IMAGES

2.1. Create the Image: the 1d Case

First, generate an image [2]. In IDIDL, we matched the window size to the image size. When using `display`, you don't need to do that; `display` matches the image size to the pre-exisging window size, or it will even generate a window for you if you don't have one open. If you want a larger image on your screen, you can define a window of your desired size and `display` will fill it. You have to be careful, though, about the effects of reducing the size (number of pixels) of an image because the interpolation process can lose information, depending on how it's done. The

[2]There's a nice 1d image of the X-ray sky, obtained by the ROSAT satellite, in `http://astro.berkeley.edu/~heiles/handouts/rass_c.fits` . Copy this file to the disk area where you are running IDL. To accomplish this, type `wget http://astro.berkeley.edu/~heiles/handouts/rass_c.fits` . This file is in a format called "fits" format, which is the standard format of most astronomical images. To read the data file into an array called `img`, it is easiest to use the IDL procedure called "readfits", which resides in the Goddard IDL library, which, in turn, is already in your IDL path. All you have to do is type `img = readfits('rass_c.fits', headerinfo)` . This returns two arrays: the image array (`img`) and information about the image (`headerinfo`); type `print, headerinfo` to see the header information. Now type `help, img` and IDL will tell you that it is a 480×240 FLOAT array.

discussion of `congrid` in IDIDL explains these concerns, which apply to `display` even though it doesn't use `congrid`.

A good way to use `display` for this particular image:

```
x=findgen(480)* (360./480.)
y=findgen(240)* (180./240.) - 90.

display, img, x, y, min=0, max=2000, out=out, $
xtit='Galactic Longitude', ytit='Galactic latitude', title= 'C-Band ROSAT Sky'
```

Some explanation:

1. `x` and `y` are the axes values. The angles run from 0 to 360 in `x` and from –90 to 90 in `y`. The first dimension of `img` must have the same number of elements as `x` while the second dimension of `img` must have the same number of elements as `y`.
2. `min` and `max` say that the minimum data value will be converted to 0 (the minimum intensity, black for a greyscale image) and the maximum to 255 (white).
3. If you wanted to make a false-color image instead of greyscale, load in your desired color table. For example, if you want to use a native IDL color table, e.g. number 5, then before calling `display` type `loadct, 5`.
4. The x, y, and image titles speak for themselves.
5. `out` is an output structure that contains information used for examining and manipulating the image; check it out. It is used directly as an input to `trp` .

In addition, there are some useful additional keywords or optional inputs for `display`. Some of the more useful include:

1. `aspect` allows you to change the shape of the image. `aspect` is the ratio of the x to the y dimensions of the output image.
2. `negative`: if you set this keyword, then the color table is reversed. That is, instead of black/white being min/max, black/white is max/min. For greyscale images to be printed on paper, or projected in a powerpoint presentation, you *almost always* want to set `negative`!
3. If you want to change the contrast of the image, you must do it yourself before calling display. For example, to increase the contrast so that weak features are highlighted, one often raises the image values to a power less than one. This power is usually called `gamma`.

2.2. 3d Images

Everything here is exactly as above, except that **(1)** your image must be a truecolor image of dimension (3, nx, ny) [i.e., 3 colors by (nx/ny) (horizontal/vertical) pixels]; and **(2)** you need to set the `true` keyword when calling `display`.

2.3. Interactively Locate Points Using `trc`

`trc` is just a wrapper that calls `tr_rdplot` with useful settings. Type

```
trc, x, y
```

Then move the cursor inside the plot window. Left clicks write and record the `(x,y)` values in the coordinate system you select (i.e., `data, device, normal`; default is `data`). A middle or left click will exit the program. Notice that the crosshairs are clipped at the axes: it's often useful to make the crosshairs extend to the edges of the window. This is accomplished by passing the `/noclip` keyword, and since we can use minimum matching when calling keywords in IDL, `/noc` will be good enough:

```
trc, x, y, /noc
```

If you don't want to store the positions of your left-clicks in `x` and `y`, then you don't need to include them in the procedure call. For more details, see the documentation of `trc` and `tr_rdplot`.

2.4. Interactively Plot 1d Image Intensities Versus x or y Using `trp`

`trp` is just a wrapper that calls `tr_profiles` with useful settings. Type

```
trp, out
```

Notice that a new window has been created. Move the cursor inside the plot window containing the displayed image. You now see the profiles in the newly created window (labeled *Profiles*). Initially, you see intensity versus the `x` data coordinates. Left click gives intensity versus the `y` data coordinates. Middle click prints the x/y values. Right click exits the program and closes the profiles window. For more, see the documentation for `trp` and `tr_profiles`.

Normally `trp` plots profiles of the original, unscaled image. However, if you have scaled the displayed image using `min` and/or `max`, and if you want to get profiles of this scaled image, type

```
trp, out, /scaled
```

Obviously, in order for all this to work, you need to have used the `OUT` keyword for `display` prior to calling `trp`.

3. ADD A COLORBAR! USING TIM ROBISHAW'S `colorbar`

Displaying an image without a colorbar is like displaying a graph with axes that have no numbers or labels. It's easy to add a colorbar with Tim's `colorbar`. This procedure does either 1d or 2d colorbars; see the images in the handout "1d2d3d: One, Two, and Three Dimensional Color Images". Here's what you do:

1. Decide where to put the colorbaar. Decide if you want a horizontal colorbar (usually on the top) or a vertical one (usually on the right hand side). There's a keyword `vertical`; set it appropriately.

2. Define `position`, a 4-element vector that specifies the normalized coordinates[3] of the corners of the colorbar. The 4 elements are in order (lowleftx, lowlefty, uprgtx, uprgty). For example, for a horizontal colorbar on the top you might choose something like `position = [ 0.1, 0.8, 0.9, 0.9]` . If you have a horizontal colorbar and you want it to extend the exact length of the image, you can get the normalized position of the left and right edges of the image from the system variable !x.window; similarly, the top and bottom are stored in !y.window. So if you wanted a colorbar that matched the width of the image and you wanted it to have a height of 10% of the height of the window, then you could make the colorbar float 5% above the top of the image by providing a position vector of:

   ```
   position = [!x.window[0],!y.window[1]+0.05,!x.window[1],!y.window[1]+0.05+0.10]
   ```

   You need enough space for the colorbar. If you havn't made a big enough window, then when you run `colorbar` it will spill over the edge of the window, in which case you might get an error like this:

   ```
   % DISPLAY: Normalized POSITION[2:3] must be less than 1.
   % Execution halted at: DISPLAY           731
   ```

3. Define the data values that correspond to the min and max of the image brightness; for the description of `display` above, these would be `min` and `max`. Set `crange=[min, max]`.

4. If you raised the data to a power (`gamma`) to change the contrast, then set `cgamma = gamma`. If not, set `cgamma=1` or leave it undefined.

5. Define the name of the displayed quantity, e.g. `xtitle='Xray intensity'`.

6. Define `position` as above.

7. Then make the colorbar:

[3]Normalized coordinates range from 0 to 1 and are the fraction of the size of the window.

```
colorbar, POSITION=position, CRANGE=crange, CGAMMA=cgamma, $
        NEGATIVE=negative, VERTICAL=vertical, XTITLE=xtitle
```

For more info on `colorbar` parameters, see its documentation.

4. SUMMARY: MAKING AN IMAGE AND ITS COLORBAR

If you want a nice image/colorbar pair, then do the following:

1. Display the image using `display`. You need to specify the size of the image and, in particular, you need to make it smaller than the PostScript window size; this leaves room for the colorbar. So you might say

```
x=findgen(480)* (360./480.)
y=findgen(240)* (180./240.) - 90.
min=0
max=2000
position= [.15, .15, .85, .85]
display, img, x, y, min=min, max=max, out=out, $
       xtit='Galactic Longitude', ytit='Galactic Latitude', $
       title= 'C-Band ROSAT Sky'
```

   N.B. If you want to annotate the image, do so here using `xyouts` .

2. Now we display the colorbar. Manipulate the `position` coordinates, using trial and error, to make it look visually good.

```
position = [0.15, 0.88, 0.85, 0.94]
xtitle=  'ROSAT counts'
colorbar, POSITION=position, CRANGE=crange, CGAMMA=cgamma, $
        NEGATIVE=negative, VERTICAL=vertical, XTITLE=xtitle
```

If you want all this on PostScript with an 8-inch square window, begin with

```
ps_ch, 'filename.ps', /defaults, /color, xsize=8, ysize=8, /inch
```

(or you can use `psopen`). Then, at the end, close the PS file with

```
ps_ch, /close
```

5. CREATING A 2d IMAGE AND COLORBAR

5.1. Using `display2d` for 2d Images

The easiest way to make a 2d image with its colorbar is with our `display_2d.pro`; this does everything with one procedure call. For an example showing how it's used, see the appendices in our tutorial "1d2d3d: One, Two, and Three Dimensional Color Images".

2d images synthesize color and intensity, so these plots explicitly require information for the two dimensions—i.e., the two images. Define

1. `c_img` and `i_img` as the color image and intensity image, respectively.
2. `xaxis, yaxis` as the (x,y) pixel values (like `x,y` for `display`).
3. `c_title` and `y_title` as the titles for the two images
4. Keyword `c_range`: a 2-element vector specifying the max and min of color used for labeling (e.g., if colors correspond to a velocity range of -10 to 10 km/s, then `c_range = [-10,10]`).
5. Keywords `intmin, intmax` as the minimum/maximum intensity values for the intensity images (like `min,max` for `display`).
6. Keyword `gamma` as the contrast parameter (the intensity image is raised to this power).
7. Keywords `xaxlbl, yaxlbl` as the x- and y-axis labels.
8. For PostScript, set keywords `ps`, `filename` and, if you want encapsulated PostScript, `encapsulated`.

There are some other keywords; you can look at the code and experiment.

5.2. The Alternative: Do It All Yourself

1. For a 2d image you need to specify the min and max for both the color image (`cmin, cmax`) and the intensity image (`imin, imax`). These min/max values are specified as 2-element vectors. So for the color image, it's `crange= [cmin, cmax]` and for the intensity image it's `irange= [imin, imax]`. Similarly, each image has its own gamma (the power to which the data numbers are raised): `cgamma` and `igamma`. It should be obvious that these gamma values should correspond to what you used when you displayed the image. The titles on the colorbar are `xtitle` (the name of the variable that represents color, e.g. velocity) and `ytitle`.
2. Finally, for a 2d color image and colorbar you are using a 256-entry colortable to define the colors. The 256 (r,g,b) intensities are represented with a 256×3 array called `colr`. If you use the physiologically-correct colortable discussed in the handout "1d2d3d: One, Two, and

Three Dimensional Color Images", then you get this array from the procedure `pseudo_ch, colr`. We assume this in our example just below; if you get `colr` in another way, you still need to `loadct,0` before calling `colorbar`.

3. Finally, invoke the colorbar procedure:

```
pseudo_ch, colr
loadct,0
colorbar, POSITION=position, RGB=colr, $
        CRANGE=crange, IRANGE=irange, CGAMMA=cgamma, IGAMMA=igamma, $
        NEGATIVE=negative, VERTICAL=vertical, $
        XTITLE=xtitle, YTITLE=ytitle
```

N.B. Any of the keywords that you can send to `plot` are also accepted by `display` and `colorbar`. As an example, sometimes it's easier to see tickmarks on an image and on a colorbar if they are pointing outwards; this could be accomplished by passing the keyword `ticklen=-0.02` to both `display` and `colorbar`.

6. USING `rgbimg` FOR 3D COLOR IMAGES

Tim's `dieplay` handles 3d images. But how do you make a colorbar for 3d images? For our solution, see our handout "1d2d3d: One, Two, and Three Dimensional Color Images" (newdimensionalcolor.ps). Unfortunately, however, we do not have a colorbar procedure that produces our 3d colorbar.

What we can offer is a clumsily-written but reliable procedure to produce a 3d image with a fancy colorbar like those of Figure 6 in the handout "1d2d3d: One, Two, and Three Dimensional Color Images". A severe restriction is that the image must have dimensions 541×541; you can use `congrid` to convert your image to this size (but see the cautions about `congrid` in in the handout "1d2d3d: One, Two, and Three Dimensional Color Images"). The procedure is called `rgbimg.pro` and it is reasonably well-documented.