Subject: Coyote Graphics Updates
Posted by David Fanning on Thu, 08 Dec 2011 15:16:05 GMT
View Forum Message <> Reply to Message

Folks,

I'm not sure what accounts for the burst in activity
in the past week (fears of having to get a real job
again, probably!), but I have changed 28 of the 41
Coyote Graphics routines in the past week!

Most of the changes have been documentation changes,
as I am switching to the IDLDOC rst method for all
new programs, and I wanted to retrofit all of my
Coyote Graphics programs so I could have a good
set of on-line documentation. There are still
niggling typos, etc., but you can find the on-line
documentation here:

http://www.idlcoyote.com/idldoc/cg/index.html

There have also been changes to some well-used programs.

cgColor -- I have modified this grandfather of all
Coyote Graphics programs to be a bit more useful.
It will now accept a three-element array (a color
triple) as input, so you can specify *exactly* what
color your want in a device and color model independent
way. It will also accept byte and integer values, and it
will treat such values as indices into the current
color table. Previously, you had to make a string of
these values first (i.e., '215', rather than just 215).
It will also complain vociferously now if you pass
it the ambiguous long integer.

Eventually, this will make specifying colors for all
Coyote Graphics routines more robust, but we see the
effects immediately in changes to cgColorbar.

cgColorBar -- The biggest change to cgColorBar is
the ability now to include "out-of-bounds" colors
as triangles on either end of the color bar. The
keyword OOB_Low and OOB_High allow you to specify the
colors you want to use and take advantage of cgColor's
new functionality so there are LOTS of options on
how you specify the colors you want to use. The shape
of the triangle can be adjusted with the OOB_Factor
keyword, for those of you who are anal about such

things.

I find I use cgColorBar very often with a handful
of colors, and I would like the colors to look like
distinct rectangles, in the manner of cgDCBar. This
involves some manipulation of keywords, that I think
a lot of people don't know how to use. So, I have added
a Discrete keyword, that just sets this up for me.

```
   cgLoadCT, 33, NColors=10
   cgColorbar, NColors=10, /Discrete
```

To add out-of-bounds colors, you do this. Note
that the OOB colors are independent of each
other. You can use one without the other.

```
   cgErase
   cgColorbar, NColors=10, /Discrete, $
     OOB_Low='white', OOB_High='Black'
```

One motivation for the color bar changes was a desire
to make a color bar that more accurately reflects the
reality of a filled contour plot.

Often, we create a filled contour plot with a color bar
like this:

```
  data = cgDemoData(2)
  cgLoadCT, 33, NColors=10, Bottom=1
  step = (Max(data) - Min(data)) / 10
  levels = Indgen(10)*step + Min(data)
  cgContour, data, Levels=levels, /Fill, $
    Position=[0.1, 0.1, 0.9, 0.825], C_Colors=Indgen(10)+1
  cgContour, data, Levels=levels, /Overplot
  cgColorbar, NColors=10, Bottom=1, $
     Range=[Min(data), Max(data)], Discrete
```

But, this gives us fairly arbitrary contour levels. We often
want the levels of our choosing, but when we do so, the last
level usually means something like "this color represents all
values above this level". In other words, we want something
that looks like this:

```
  data = cgDemoData(2)
  cgLoadCT, 33, NColors=10, Bottom=1
  levels = Indgen(10)*150
  cgContour, data, Levels=levels, /Fill, $
    Position=[0.1, 0.1, 0.9, 0.825], C_Colors=Indgen(10)+1
  cgContour, data, Levels=levels, /Overplot
  cgColorbar, NColors=9, Bottom=1, OOB_High=10, $
     Range=[Min(levels), Max(levels)], /Discrete
```

To make this even easier to do, I have added a new OLEVELS
keyword to cgContour, so that you can, if you need to, fetch
the contour levels that the program actually uses. So now,
you can do the same thing, but using NLevels in the
cgContour call in the usual way:

```
  data = cgDemoData(2)
  cgLoadCT, 33, NColors=10, Bottom=1
  cgContour, data, NLevels=10, /Fill, OLevels=levels, $
    Position=[0.1, 0.1, 0.9, 0.825], C_Colors=Indgen(10)+1
  cgContour, data, Levels=levels, /Overplot
  cgColorbar, NColors=9, Bottom=1, OOB_High=10, $
     Range=[Min(levels), Max(levels)], /Discrete
```

cgPS2PDF -- The functionality that caused the biggest change
to Coyote Graphics programs was the addition of the
cgPS2PDF program that allows me to create PDF files
in a machine-independent way from PostScript intermediary
files.

I know people use a lot of different routines to do this,
and I have tried to retain that kind of flexibility in this
program. I'm not out to reinvent the wheel, I just really
want the ability to make PDF files from within cgWindow! :-)

This program will use Ghostscript on Windows and UNIX machines,
and pstopdf on Macs. If you don't want to use the "gs"
command on UNIX machines, you can choose the command you
do want to use. For example, the "epstopdf" command probably
works better with encapsulated PostScript files than does
the "gs" command. You can make that substitution in the
program.

A number of Coyote Graphics programs changed to accommodate
this new functionality. Among them are cgWindow, cgControl,
cgWindow_SetDefs, cgWindow_GetDefs, and PS_End. I wouldn't be
surprised if there were others as well.

You can find all of these changes in the latest Coyote Library
distribution:

    http://www.idlcoyote.com/programs/zip_files/coyoteprograms.z ip

Or, using Subversion at:

http://idl-coyote.googlecode.com/svn/tags/coyote/coyote_libr ary_1.4

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")