Subject: Loading Color Tables on a 24-Bit Display
Posted by davidf on Mon, 14 Apr 1997 07:00:00 GMT
View Forum Message <> Reply to Message

Hi Folks,

A number of people have asked me recently about loading color tables on a 24-bit display. Most of the time their concern is that they are writing code that will sometimes be run on an 8-bit display and sometimes on a 24-bit display, and they want the code to behave identically in both cases. Specifically, they want to see their graphics displays updated immediately when they load a new color table from within the program on a 24-bit display, just like it does when they work on an 8-bit display.

Unfortunately, this doesn't happen in IDL. Most people who run IDL on a 24-bit display for the first time are surprised to learn that graphic displays and images have to be redisplayed after a color table is loaded for the new colors to take effect. This is simply a case of using the 24-bit DIRECT color model, in which colors are specified directly, instead of the more familiar 8-bit INDEXED color model in which colors are tied or indexed to a value in a color lookup table.

It is this requirement to redisplay the graphic or image after colors are loaded that makes it difficult to work with IDL in a 24-bit mode.

Can anything be done about it?

Well, yes, sometimes, but especially if you write widget programs and follow a few simple rules for how you display graphics output in your widget draw windows. I've written a couple of simple programs to show you how. These programs work identically on 8-bit, 16-bit, and 24-bit displays and in IDL 4 and IDL 5 (as non-blocking widgets). Color tables can be loaded and the graphic display updated immediately.

The most important program is named XLOAD. It is similar to the familiar XLOADCT and is a modification of one of my other color table programs named XCOLORS. The most important modification to XLOAD has been the addition of a NodifyID keyword, which can be used to pass into XLOAD the identifiers of widgets that should be notified when XLOAD loads new colors into the color table. In fact, the notification consists of an {XLOAD_COLORS} event structure

passed to the specified widget.

The event handler for the specified widget (which is usually a draw widget) is responsible for redisplaying the appropriate graphic in the draw widget window. (This is similar--if not identical--to what must be done for resizeable graphics windows.) You can see how this is done in the second program, named PROCESS, with makes use of the XLOAD program to load its color tables.

I have created the draw widget in the PROCESS program like this:

```
  drawID = Widget_Draw(drawbase, XSize=xsize, YSize=ysize, $
    Event_Pro='Process_Draw_Events')
```

I stored the draw widget's identifier (drawID) in the "info" structure where I can get at it from any event handler. In the event handler in which I call XLOAD, I call the program like this:

```
  XLOAD, Group=event.top, NColors=info.ncolors, $
    Bottom=info.bottom, NotifyID=[info.drawID, event.top]
```

Notice that the NotifyID keyword requires two values. The first is the identifier of the widget that the notification event will be sent to and the second is the identifier of the widget at the top of the hierarchy to which the first widget belongs. This is a convenience and a convention for my widget programs, since I always use the top-level base as a pointer to the "info" structure. (The NotifyID keyword can actually take a 2-by-n array of widget values, defined as above, so you can notify many widgets if you like.)

When color tables are loaded in XLOAD, an event structure is created and sent to the specified widget with the SEND_EVENT keyword to WIDGET_CONTROL. The code that does this looks like this:

```
    ; Are there widgets to notify?

  s = SIZE(info.notifyID)
  IF s(0) EQ 1 THEN count = 0 ELSE count = s(2)-1
  FOR j=0,count DO BEGIN
    colorEvent = {XLOAD_COLORS, $
                  ID:info.notifyID(0,j), $
                  TOP:info.notifyID(1,j), $
                  HANDLER:0L, $
```

```
                    R:r, $
                    G:g, $
                    B:b }
    IF Widget_Info(info.notifyID(0,j), /Valid_ID) THEN $
      Widget_Control, info.notifyID(0,j), Send_Event=colorEvent
  ENDFOR
```

Notice that the red, green, and blue color vectors that make up the current color table are passed in this event structure. This information is often desired and used by the event handler that receives this event.

In this case, the event handler assigned to the draw widget when I created the draw widget is PROCESS_DRAW_EVENTS. The important portion of that event handler is this:

```
  IF thisEvent EQ 'XLOAD_COLORS' THEN BEGIN
    WSET, info.wid
    ok = Execute(info.command)
    info.r = event.r(info.bottom:info.ncolors-1+info.bottom)
    info.g = event.g(info.bottom:info.ncolors-1+info.bottom)
    info.b = event.b(info.bottom:info.ncolors-1+info.bottom)
  ENDIF
```

The graphic is redisplayed, in this case, by executing the "command" or "action" that is stored in the command field of the "info" structure with the EXECUTE command. (In IDL 5 this "action" might more easily be accomplished by evoking the "draw method" of the graphic object.) I also update the color vectors that are stored in the info structure of the program. This guarantees that my program always knows about its "current" colors. (A self-imposed requirement for non-blocking widgets in IDL 5).

You can download the PROCESS and XLOAD programs via the Example IDL Programs section of my web page, located at:

  http://www.dfanning.com

Please sent your comments or questions directly to me at:

  davidf@dfanning.com

I do expect this program to change a little as I explore its usefulness over the next few weeks.

COMMERCIAL APPEAL ----------

I spend quite a bit of time writing IDL programs in a way
that people can understand and learn from. I also try to
post useful advice on this newsgroup. Many of my friends on
this newsgroup have told me I am crazy to give so much of
this work away for free. They are probably right. But I
like to give it way and I want to continue to do so. :-)

If you like these programs and find them useful, I would very
much appreciate it if you would keep me in mind when you need
someone to help with your next IDL programming project or
when you think you are ready for an IDL programming class.

If you don't have any work for me, an e-mail--even though it
doesn't feed the kids--does lift my spirits and keeps me
going. :-)

Enjoy!

David

David Fanning, Ph.D.
Fanning Software Consulting
2642 Bradbury Court, Fort Collins, CO 80521
Phone: 970-221-0438 Fax: 970-221-4762
E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com
