Subject: Re: Color Frustration
Posted by J.D. Smith on Sun, 02 Mar 1997 08:00:00 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> Right-oh. Get rid of them. Completely. Do away with them. This is
> the thing that is frying your goose (er, something!).
>
> No way you should have TVLCT, r, g, b, /Get *AND* a common
> block. Stuff and nonsense. One or the other, my man. And I say,
> chuck the block.
>
> All this copying to and fro is doing you in. Putting you under, so
> to speak. Casting stars in your eyes, if you catch my drift. Colors
> floating about.
>
> You want colors, go *get* them:
>
> TVLCT, r, g, b, /GET
>
> You want to load them, load them:
>
> TVLCT, r, g, b
>
> Or, what I prefer:
>
> LOADCT, 5, NColors=200, Bottom=10 ; or whatever,
>
> But don't be messin' around with no *common* block. No, sir, you're
> better than that! Steer clear of that riff-raft.
>
>> Well, if you made it this far, it's a small miracle,and a testament to
> your IDL
>> fanatasism.
>
> Do you believe I am not only reading this, but responding after a six-pack
> and working 14 hours on my IDL manual today. I have to *seriously* get a life!
>
>> And yes, David, it is a "fantastic story", but aren't all the interesting
> ones?
>
> Yes, indeed. And this one has cheered me up more than most. :-)
> I'll buy the beer next time I see you , JD!
>
> Cheers!
>
> David

Well, I have finally figured out what was wrong, and I should have suspected it from the beginning. It's not the use of the color common block, which, as it turns out, I actually do need, in order to set my plotting color into the "original" color vectors (though I found a cleaner way to implement it). This allows the user to pick any color table he wants,and stretch it and chop it to his hearts content, without affecting my green over-plotting color. More on this later.

The real culprit is the XOR graphics_function mode. I was temporarily thrown off the trail by my unorthodox use of the colors common block.

The scenario is this: A zoom rubber band is drawn with the XOR mode, but since I hadn't anticipated all uses of the draw widget when I wrote it, I tried to optimize things by setting XOR when the user first clicked (initiating the zoom box process), and then unsetting it only after the user released, with arbitrarily many zoom boxes drawn (with XOR) in between. The DEVICE,set_graphics call is actually pretty slow, and so this was speeding things up a good bit. The problem is, if you use this widget together with any others that do any drawing (which I am in this case), you're going to run into troubles. It sounds obvious, but it is somewhat subtle, since what is critical is the order in which events arrive down the pipeline. Since this is not always apparent, I would reccommend that if you use the XOR mode, set it back to copy *immediately* after every use, e.g., drawing your selection rubber band. This does incur a performance penalty, but ensures you won't run into these difficulties. This is the only guaranteed way to ensure that the XOR mode (or any other mode for that matter) doesn't "leak" out into the rest of your widget environment. And the wierd behavior of graphics_function leakage is somewhat unpredictable, since the ordering of events is not always obvious. What seems like a perfectly self-contained application of XOR may not remain so when widget are teamed up!

A side note on the use of the colors common block: The difficulty is that setting green with:

```
tvlct,0B,255B,0B,!D.N_COLORS-1 ;load green into top color
```

doesn't affect the common block variables (only the device's color tables). Any subsequent call to stretch, which uses the colors block original variables (r_orig etc.) to construct the current ones (r_curr etc.), will obliterate your green, *unless* it is set into the original variables. In addition, a call like

```
LOADCT, 5, NColors=200, Bottom=10
```

will similarly lose the green, since it sets the variables with:

```
r_orig(cbot) = r   ;set a portion of r_orig with loaded table
g_orig(cbot) = g
b_orig(cbot) = b
r_curr = r_orig
g_curr = g_orig
b_curr = b_orig
tvlct,r, g, b, cbot
```

As you can see, even if r,g, and b are only 10 elements vectors, the current color vectors will be overwritten with the *entire* original ones. The only option is to set the green into the original, and then reset the originals after the widget dies, and the only way to accomplish this is to deal with the common block directly.

Of course, the other option is to maintain a single colormap during your widget's lifetime, but this limits the user's ability, in my application, to explore that data.

A further application of this technique, which I am currently putting some thought into, would allow the total colormap to be chopped up into as many smaller colormaps as you want, each of which can be stretched and manipulated (e.g. loading new color tables into) independent of the rest. Then you could have two (or more) images with different colormaps, and a few colors for overplotting, and a colormap tool that could use the active image to determine which sub-colormap to display and edit. Cool, eh?
