Subject: Re: A defense of decomposed color
Posted by David Fanning on Wed, 01 Nov 2006 02:34:49 GMT
View Forum Message <> Reply to Message

JD Smith writes:

> I'm a believer in the power of those tools, but was more interested in
> general suggestions for people who want to leverage this
> all-things-to-all-people color model in their own code.

Oh, well, then call the good folks at ITTVIS and tell them
Matlab out-sells IDL because the good folks there know that
no one wants to fool around with this stuff. That might
leverage them a little bit. :-)

```
>>  Here are just a few of the advantages of using TVIMAGE or IMGDISP:
>>
>>     1. Don't have to worry what color mode you are in, ever. They do
>>        the *right* thing to get color images correct. They can tell
>>        the difference between a Windows and UNIX machine.
>
>  For those of us with our own display code, can you summarize *how*
>  they do the right thing?  For instance, if you have an 8-bit display,
>  where you're forced to use color tables, how can they anticipate how
>  many "system" colors a program may end up needing?
```

I'm not sure exactly what you mean by "system" colors in this
context, but I presume you mean you want to use some colors from
the color table for the image (maybe all of them), and you want
to use others for drawing colors. The colors loaded at the moment
are the "system" colors. If this is not what you mean, please
let me know.

First of all, TVIMAGE and IMGDISP don't know anything about
colors. They assume (usually erroneously) that you know what
you are doing with colors and they will just go along with
whatever you decide. So you can load your system colors in
whatever way makes sense to you. I typically use all 256
colors for my image.

So I load the color table, if I have an 8-bit image I want
to display. I must do this, ALWAYS, right before I display
the image that uses the colors. (This is why an image object
is nice, because I can create an "image" that has a color
table already associated with the image data. When I "draw"
the image, it always loads its color table first.)

All TVIMAGE and IMGDISP do is figure out whether I have

a 2D image or a 3D image, and if I have a 3D image, which
dimension is a 3. They know that to display a 2D image
property, I have to use indexed color to display it. So
they save the current decomposed state, and then set the
display to 0:

```
   Device, Get_Decomposed=theState, Decomposed=0
```

If I have a 3D image, then they either have to display
the image with color decomposition turned on:

```
   Device, Get_Decomposed=theState, Decomposed=1
```

Or, if this is an 8-bit device, they have to COLOR_QUAN the
24-bit image and display the resulting 2D image after loading
the color table vectors that are returned from COLOR_QUAN
and turning color decomposition off.

When they are finished, they set the color model back to
whatever it was when you entered the program.

FSC_COLOR does much the same thing. If you have color
decomposition turned on, no colors have to be loaded
in the color table, and you specify the color "directly".
But if it is turned off, you have to load the color
somewhere. If you don't tell me where you would like
to load it, I load it at a unique location in the color
table. In either case, I return to you the location where
I loaded it.

If I load a color, this "pollutes" your image color
table (probably). This is why you have to load your
image colors again before you display your image.
Of course, judicious use of your color table space
allows you to share the color table with an image
and drawing colors, just the way we had to do this
previously in the 8-bit world. And, in fact, the
PRINTER device *makes* you share your color table
in just this way, because you can only load the colors
in the PRINTER device once. (Whereas you can load
colors in PostScript or the Z-buffer as many times
as you like.)

Naturally, none of this would work on a real 8-bit
display, because as soon as I loaded my drawing
color, my image color at that location would change
too. And even though you assert a 95% penetration
of 24-bit color displays, JD, I would claim an

even higher percentage. I haven't run into a actual
8-bit display in many years.

```
>>     2. FSC_COLOR works in a color model independent way. If you are
>>        on an 8-bit device, FSC_COLOR loads the color in the color table
>>        (you can tell it where or it will choose a location) and it will
>>        return the location (index). If you are on a 24-bit device,
>>        FSC_COLOR will do the 'fe458f'xL thing for you, not bother to
>>        load a color, and you will get the SAME COLOR you get on an
>>        8-bit device. No ugly code to puzzle over. The following
>>        code works on your display and in PostScript without caring
>>        what color model you are using. Plus, you can read it and
>>        have a good idea of what colors you SHOULD be seeing!
>
>  Does the postscript device demand color-table style colors?  That to me
>  would be a big drawback of decomposed color (unless you consistently use
>  FSC_COLOR/etc.).
```

The PostScript device conveniently allows you to load colors
in the color table at any time while you are in the device,
so I can draw a line with a red color loaded at index 10, then
change index 10 to a blue color and draw something else. This is
NOT possible in the PRINTER device. There, if I want two drawing
colors, I must load both into the color table at different indices
and the colors can be loaded only once (when I select the PRINTER
device).

You DO have to be a bit more careful about the way you load
colors and the order you draw things if you plan to use the
PostScript device, but this skill is quickly learned.

```
>>  Bottom line. Don't worry about what color model you are using. Set
>>  yourself up in color decomposed mode, use the proper tools to work in
>>  that mode, put your feet up, and never think about it again. Problem
>>  solved! :-)
>
>  That's a good approach for interactive users on the command line, but
>  doesn't really constitute a complete solution for applications, which
>  may need to optimize color usage for various types of data.  If the
>  actual code every time you want to draw a plot element on top of some
>  image is something like:
>
>   device,get_decomposed=gd
>   device,/decomposed
>   plot,x,y,COLOR='fe458f'x
>   device,decomposed=gd
>
>  not only is this inefficient (taking about 5ms extra for the decompose
```

> fiddling -- not so bad, what if code which is called 100's of times
> after a motion event?), but it's not maintainable either.

Well, it's ugly as sin, too! :-)

I've never noticed these kinds of color manipulations affecting
speed, even with motion events. Maybe my machine is faster than
yours, but I really, really doubt it.

> Here's my ideal scenario:
>
> 1. I can use all 256 colors for colormap drawing of images.
> 2. I can use any additional color as 'ffaabb'x.
> 3. Will degrade gracefully in 8bit color.
> 4. Won't require elaborate setup and takedown of the color space
>   everytime I want to put something to screen.
>
> I think that last point is what has kept me in color-table land for so
> long. I know the combo of TVIMAGE/FSC_COLOR does this for you, but
> I'm looking for the secret behind the sauce.

Well, of course, I scramble the code in all the programs I post
on my web page, but I think Liam's code is plain text. :-)

>> P.S. By the way, an image object that displays itself with TVIMAGE and
>> contains its own color table can display itself correctly anywhere and
>> anytime. You don't even have to remember to load the colors anymore! :-)
>
> Hmmm... is that because TVIMAGE keeps track of it for you?

No, because the object is bright enough to do it for me. It
just won't display the image without first loading the associated
color table.

Cheers,

David

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