Subject: Re: A defense of decomposed color
Posted by David Fanning on Tue, 31 Oct 2006 04:17:23 GMT
View Forum Message <> Reply to Message

JD Smith writes:

> I wonder if those of you using decomposed color can persuade me of its
> utility. Though color tables are perfect for image visualization,
> they are wanting for "system" colors for plot symbols, overlays, etc.
> It's frustrating to keep track of them, and different apps have
> different conventions, and can step on each other's feet, causing
> various undesirable effects.
>
> I presume the reason many of us still use undecomposed color is from
> the 8bit heritage, when there was no such thing as decomposed color.
> Do '00FFFF'x-loving people simply assume everyone has a device capable
> of interpreting 24bit, decomposed color (probably about 95% true,
> these days)? How do you handle switching back and forth from
> decomposed (for plot symbols, say) to indexed (for displaying images)?
> Do you find it really solves the headaches associated with saving a
> few colors for drawing in high indices, vs. the added juggling needed
> to switch back and forth among decomposed and non-decomposed color,
> etc.? What happens if you switch to decomposed color on an 8-bit
> display system?
>
> I'm ready to come around to embracing direct color specification with no
> color table intermediary, but I think I need a bit of persuasion.

Two words, JD: TVIMAGE and FSC_COLOR.

The point of using decomposed color is that you can load your image color tables, use all 256 colors all the time, and *still* use any color you like for plots and annotation, WITHOUT HAVING TO SWITCH ANYTHING. At least you can if you use TVIMAGE (or, alternatively, Liam's IMGDISP) and FSC_COLOR. I can't remember the last time I switched color models, and I haven't known (or cared) what color model I've been using for a least the last five years.

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.

2. Don't have to worry about how to set the TRUE keyword with 24-bit images. They can tell the difference between an 8-bit image and a 24-bit image. Forget about the TRUE keyword.

3. Can set keywords that preserve the aspect ratio of your image.

4. Can "erase" the display before they draw, similar to other IDL graphics commands.

5. Images can easily be "positioned" in windows with the POSITION keyword, in the same way other IDL graphics commands are positioned, facilitating combining images with other graphics commands. (Adding axes, contour plots, etc.)

6. Displaying multiple images in windows is easy with !P.MULTI.

7. Can be used as a "smart" TV command. For example, just set the TV keyword with TVIMAGE and it acts as dumb as it used to work, except that it gets your image colors right!

8. Works correctly on your display (8-bit or 24-bit) and in all other graphics devices, too, including PostScript.

9. Image "positioning and sizing" is done the same way no matter if you are displaying your image on the display or in PostScript. No more worry about XSIZE and YSIZE keywords!

Here are a few of the advantages of using FSC_COLOR:

1. You have a palette of 104 "named" colors, including all your system colors. If you don't like the ones I provide, FSC_COLOR can read a color file with your own choices.

   Do see your color selection, type this:

   IDL> color = FSC_COLOR(/Select)

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!

   Plot, data, Background=FSC_Color('ivory'), Color=FSC_Color('navy')
   OPlot, data, Color=FSC_Color('red')

   (You see an obvious advantage of using decomposed color here,

since your image color tables never get corrupted by drawing colors if you don't have to load drawing colors in the color table.)

While I'm thinking about it, get TVREAD, too. That is the counterpart to TVIMAGE. What TVIMAGE does to get image ON your display, TVREAD does to get them OFF your display and into PNG, JPEG, TIFF, and BMP files. No more having to worry about what color model you are using and whether you are on a Windows or UNIX machine (they handle colors differently). TVREAD also works properly with 8-bit devices, such as the Z-buffer.

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! :-)

Cheers,

David

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! :-)

--
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.")

Subject: Re: A defense of decomposed color
Posted by liamgumley on Tue, 31 Oct 2006 15:03:58 GMT
View Forum Message <> Reply to Message

JD Smith wrote:
> I wonder if those of you using decomposed color can persuade me of its
> utility. Though color tables are perfect for image visualization,
> they are wanting for "system" colors for plot symbols, overlays, etc.
> It's frustrating to keep track of them, and different apps have
> different conventions, and can step on each other's feet, causing
> various undesirable effects.

Put it this way: Try asking a Matlab user if they know whether they are using a 24-bit or 8-bit display, or if Matlab is running in decomposed

or undecomposed color mode. They'll probably say "Huh?".

Cheers,
Liam.
Practical IDL Programming
http://www.gumley.com/

Subject: Re: A defense of decomposed color
Posted by David Fanning on Tue, 31 Oct 2006 15:39:39 GMT
View Forum Message <> Reply to Message

Liam Gumley writes:

> Put it this way: Try asking a Matlab user if they know whether they are
> using a 24-bit or 8-bit display, or if Matlab is running in decomposed
> or undecomposed color mode. They'll probably say "Huh?".

Exactly! But, apparently, the marketing demographics
suggest more iTool users than people who use TV.
Go figure...

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.")

Subject: Re: A defense of decomposed color
Posted by JD Smith on Tue, 31 Oct 2006 20:33:38 GMT
View Forum Message <> Reply to Message

On Mon, 30 Oct 2006 21:17:23 -0700, David Fanning wrote:

> JD Smith writes:
>
>> I wonder if those of you using decomposed color can persuade me of its
>> utility. Though color tables are perfect for image visualization,
>> they are wanting for "system" colors for plot symbols, overlays, etc.
>> It's frustrating to keep track of them, and different apps have
>> different conventions, and can step on each other's feet, causing
>> various undesirable effects.
>>

> Two words, JD: TVIMAGE and FSC_COLOR.
>
> The point of using decomposed color is that you can load your
> image color tables, use all 256 colors all the time, and *still*
> use any color you like for plots and annotation, WITHOUT HAVING
> TO SWITCH ANYTHING.

Thanks for your thoughts, David. That's the main attraction to me.
You *do* have to switch the decomposed state, but if it's handled
transparently, no one is the wiser.

> At least you can if you use TVIMAGE (or, alternatively, Liam's
> IMGDISP) and FSC_COLOR. I can't remember the last time I switched
> color models, and I haven't known (or cared) what color model I've
> been using for a least the last five years.

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.

> 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?

> 8. Works correctly on your display (8-bit or 24-bit) and in
> all other graphics devices, too, including PostScript.
>
> 9. Image "positioning and sizing" is done the same way no matter
> if you are displaying your image on the display or in PostScript.
> No more worry about XSIZE and YSIZE keywords!
>
> Here are a few of the advantages of using FSC_COLOR:
>
> 1. You have a palette of 104 "named" colors, including all your
> system colors. If you don't like the ones I provide, FSC_COLOR
> can read a color file with your own choices.
>
> Do see your color selection, type this:
>
> IDL> color = FSC_COLOR(/Select)
>

> 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.).

> While I'm thinking about it, get TVREAD, too. That is the counterpart to
> TVIMAGE. What TVIMAGE does to get image ON your display, TVREAD does to
> get them OFF your display and into PNG, JPEG, TIFF, and BMP files. No
> more having to worry about what color model you are using and whether
> you are on a Windows or UNIX machine (they handle colors differently).
> TVREAD also works properly with 8-bit devices, such as the Z-buffer.
>
> 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. 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.

> 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?

JD

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.")

Subject: Re: A defense of decomposed color
Posted by fburton on Thu, 02 Nov 2006 14:21:13 GMT
View Forum Message <> Reply to Message

In article <MPG.1fb1b5ace9585abe989d83@news.frii.com>,
David Fanning <news@dfanning.com> wrote:
> 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. :-)

MATLAB's color handling is pretty screwed up too, imho (though
for slightly different reasons). It shouldn't be necessary to
fiddle with colormaps in order to e.g. plot two contour maps
with different color scales in the same figure, just because
a figure can only have one colormap.

Francis

Subject: Re: A defense of decomposed color
Posted by David Fanning on Thu, 02 Nov 2006 14:39:19 GMT
View Forum Message <> Reply to Message

Francis Burton writes:

> MATLAB's color handling is pretty screwed up too, imho (though
> for slightly different reasons). It shouldn't be necessary to
> fiddle with colormaps in order to e.g. plot two contour maps
> with different color scales in the same figure, just because
> a figure can only have one colormap.

Ah, well, there you go. This is inevitably the age-old
problem of simplicity verses functionality. Very, very
difficult to have both, as anyone who has worked with
a Microsoft application can tell you. As someone who

has had 20 years to learn IDL's quirks, I appreciate the functionality. But as someone who tries to teach people how to use IDL, I am painfully aware of how confusing it can be to do something as "simple" as get decent PostScript output, to cite just one recent example.

My only point is that a more intelligent TV command seems infinitely more useful to me than yet another iTool, and I should think it would do a LOT more to increase sales. But I am also sure this is a minority view among the people who make these decisions.

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.")

Subject: Re: A defense of decomposed color
Posted by JD Smith on Mon, 06 Nov 2006 17:26:43 GMT
View Forum Message <> Reply to Message

Thanks for all the thoughts, David. I hadn't appreciated the subtle differences between postscript and printer devices w.r.t. color tables. I suppose I am struggling with the concept because of the typical "free-form" design of applications I work on. They might enter into a small function to draw a specific overlay polygon many different ways, and this function might be entered 100's of times for a given notional redraw (tied to a motion event, even). A single 5us penalty compared to drawing a large image is utterly negligible. However, I just tested a simple plots call, and wrapping it in "device,decomposed=1,get_decomposed=gd & ... & device,decomposed=gd" increases the time required by roughly a factor of 5. If you need to perform say ten thousand of these for a given screen redraw (even if sent to a double buffer), it would be the difference between updates occuring 100 times/s or 20 times/s (which would be noticeable). Now, in reality, your budget may well be dominated by other overheads in all sensible cases, but it just doesn't feel right to toggle the drawing mode back and forth many thousands of times per second.

Though color tables are stone-age tools, now I can simply:

a) Set up a color table with N image colors and N_max-N "system" or

plotting colors.

b) Reload that color table on enter events.

c) Plot and/or display whenever, wherever, without having to futz the color model beforehand.

Obviously, this requires a device,decomposed=0 in the startup file (which is simply documented), but it saves considerable time and trouble. The drawback is, of course, different tools have different conventions, and you always end up with conflicts.

So it appears there is no ideal solution. It also appears the IDL color model is somewhat broken. Ideally, there would be no decomposed state, and IDL primitives would "do the right thing" with the data given them. Is this how Matlab works?

JD
