Subject: Constructing Color Tables in IDL
Posted by davidf on Sat, 15 Feb 1997 08:00:00 GMT
View Forum Message <> Reply to Message

Folks,

Several people have asked me this week how to construct a color table. I've taken this to mean (perhaps erroneously) that there is general interest in the subject. :-)

So here is how I might go about it. I'll construct a simple color table first, then a more complicated one to give you the general idea of how even very complex color tables might be constructed.

First, suppose you want a color table that goes from yellow in the first color index to red in the last color index. And suppose further that you want to have 200 colors or shades in the color table. You know that the color yellow is represented by the RGB color triple (255,255,0) and that the color red is represented by the RGB triple (255,0,0).

You know also that to create a color table you need three vectors, representing the red, green, and blue color values of the colors that make up the color table. The trick, therefore, is to know how to make up those color vectors. Another way to say this is that we need to know the RGB values of all the colors in a smooth progression from yellow to red.

What would constitute a smooth progression of colors? Well, clearly the red vector values need to go from the starting red value in the yellow color to the ending red value in the red color. The same can be said for the green vector values and the blue vector values. In fact, we can write a general expression to go from any number to some other number in a arbitray number of steps. The expression looks like this:

```
vector = beginNum + (endNum - beginNum) * scaleFactor
```

where we have to define the beginning number, the ending number, and the scale factor (which will depend upon the number of steps we want to take).

Suppose we define these quantities like this:

```
beginNum = 10.0
endNum = 20.0
```

```
steps = 5
scaleFactor = FINDGEN(steps) / (steps - 1)
```

Then, using the equation above, we print:

```
PRINT, beginNum + (endNum - beginNum) * scaleFactor
```

We get:

```
    10.0000      12.5000      15.0000      17.5000      20.0000
```

So far, so good. Let's apply this to our color table problem.
We want the red vector value to go from 255 (the red value in
the yellow color) to 255 (the red value in the red color). The
green vector needs to go from 255 to 0. The blue vector needs to go
from 0 to 0.

Pretty simple. We probably don't even need to apply the
algorithm for the red or blue vectors. We can simply write:

```
steps = 200
redVector = REPLICATE(255, steps)
blueVector = REPLICATE(0, steps)
```

The green vector (according to the algorithm) is:

```
scaleFactor = FINDGEN(steps) / (steps - 1)
beginNum = 255
endNum = 0
greenVector = beginNum + (endNum - beginNum) * scaleFactor
```

Alright, now load these color vectors, and there you have it,
a color table smoothly progressing from yellow to red!

```
TVLCT, redVector, greenVector, blueVector
```

Well, how about a more complicated example. Suppose you want
a color table (still 200 colors) that goes from yellow
to red, but you want it go through a series of blue colors
in the middle of the table.

You can divide this into two problems similar to the first
example. In other words, in 100 steps go from yellow (255,255,0)
to blue (0,0,255), and then in 100 more steps go from blue
to red (255,0,0). Your code might look like this:

```
steps = 100
scaleFactor = FINDGEN(steps) / (steps - 1)
```

```
   ; Do first 100 colors (yellow to blue).

      ; Red vector: 255 -> 0
   redVector = 255 + (0 - 255) * scaleFactor

      ; Green vector: 255 -> 0
   greenVector = 255 + (0 - 255) * scaleFactor

      ; Blue vector: 0 -> 255
   blueVector = 0 + (255 - 0) * scaleFactor

   ; Do second 100 colors (blue to red).

      ; Red vector: 0 -> 255
   redVector = [redVector, 0 + (255 - 0) * scaleFactor]

      ; Green vector: 0 -> 0
   greenVector = [greenVector, REPLICATE(0, steps)]

       ; Blue vector: 255 -> 0
   blueVector = [blueVector, 255 + (0 - 255) * scaleFactor]
```

Finally, load your new color table:

```
   TVLCT, redVector, greenVector, blueVector
```

I think you can easily see how to extend this concept
to interpolating colors between any starting and ending
color triples.

By the way, you don't necessarily have to write your own
code to do this (although if you do you will have the
advantage of understanding what you are doing) because
this is exactly what XPALETTE allows you to do when
you interpolate between two colors that you have
marked by clicking on them with the cursor!

Cheers!

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

Subject: Re: Constructing Color Tables in IDL
Posted by Kirt Schaper on Wed, 19 Feb 1997 08:00:00 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> Folks,
>
> Several people have asked me this week how to construct a
> color table. I've taken this to mean (perhaps erroneously)
> that there is general interest in the subject. :-)
>
> So here is how I might go about it. I'll construct a simple
> color table first, then a more complicated one to give you
> the general idea of how even very complex color tables might
> be constructed.
>
> First, suppose you want a color table that goes from yellow
> in the first color index to red in the last color index. And
> suppose further that you want to have 200 colors or shades
> in the color table. You know that the color yellow is
> represented by the RGB color triple (255,255,0) and that
> the color red is represented by the RGB triple (255,0,0).
>
> You know also that to create a color table you need three
> vectors, representing the red, green, and blue color values
> of the colors that make up the color table. The trick, therefore,
> is to know how to make up those color vectors. Another way
> to say this is that we need to know the RGB values of all
> the colors in a smooth progression from yellow to red.

[rgb implementation deleted]

Why not deal directly with the colors in the natural
hue/liteness/saturation space? That is, convert color1 to hls1,
convert color2 to hls2, and linearly interpolate between hue1
and hue2, converting each resulting hls to rgb space.

```
; start out with rgb values for red [255,0,0]
IDL> rr = 255   &   rg = 0   &   rb = 0
; convert to hls space
IDL> color_convert,rr,rg,rb,rh,rl,rs,/rgb_hls
IDL> print,rh,rl,rs
      0.00000     0.500000      1.00000
```

```
; convert ending rgb values (for yellow) to hls space
IDL> yr = 255   &   yg = 255   &   yb = 0
IDL> color_convert,yr,yg,yb,yh,yl,ys,/rgb_hls
IDL> print,yh,yl,ys
      60.0000     0.500000      1.00000

; (note that these values are fully saturated, and half way between
; white and black wrt liteness)

; now interpolate between the hues, converting each hls value to rgb
IDL> steps = 5
IDL> .run
- for h=rh,yh,(yh-rh)/(steps-1) do begin
-    color_convert,h,0.5,1.0, r,g,b, /hls_rgb
-    print,r,g,b
- endfor
- end
 255   0   0
 255  63   0
 255 127   0
 255 191   0
 255 255   0
```

This guarantees that the colors are maximally spread in hue space.
Unfortunately, hue space is not perceptual space (or indeed
monitor display space), so the colors may NOT be maximally
separate wrt a human observer on a given monitor.

Kirt Schaper | Bell: (612) 725-2000 x4791
PET Center (11P) | net: kirt@pet.med.va.gov
VA Medical Center | FAX: (612) 725-2068
MPLS, MN 55417 | URL: http://pet.med.va.gov:8080