Subject: Re: 24 bit colors in IDL
Posted by philip aldis on Fri, 06 Nov 1998 08:00:00 GMT
View Forum Message <> Reply to Message

Richard Penrose wrote:

> I am working on a piece of software written in IDL that has always
> assumed the color resolution to be 256 colours. I would like to be able
> to use the same piece of software on a machine which has a flashy
> graphics card and only allows a colour resolution of 24 bits (True
> Colour).
>
> At the moment when I change the colour resolution it messes up all of
> the software's colours. The piece of code that we use to set up the
> colours is as follows:
>
> -----code deleted----------

>
> Now it seemed obvious to me that to upgrade to 24 bit colours all I
> needed to do was to
> multiply each element of the iaR, iaG, iaB arrays by 65536 and subtract
> 1. Disappointingly this does not seem to work!
>
> Can anyone help.
>
> Cheers Richard

Before I start, check out, http://www.dfanning.com/tips/colors24.html, which has some tips on 24 bit colour.
I don't know whether you know the difference between 24 bit colour and ordinary colour, but I will attempt to give a brief explanation. With the sort of colour that you have been using in the past, colours are not specified as the exact colour, i.e. a pixel does not say, I have 255 parts red, 50 parts green and 70 parts blue. Instead it has a colour index which ranges from 0 to 255. This number is referring to a colour map, a table which has 256 colours and for each colour index the table gives the RGB triple to describe that colour. The pixel is impervious to its actual colour, it only knows its index. If you made every index in the colour map read RED, 255, GREEN, 0, BLUE, 0 (by using tvlct)and then displayed an image, it would be entirely red. If you then loaded all GREENS into the colour map, the image displayed on your screen would become all green. This is because, as I said, the pixels don't know what colour they are, they simply know what index in the colour map tells them what colour to be, so when you change the actual values in the colour map, by loading in all greens, the pixels also change colour.
Here lies the fundamental difference between 24 bit colour, and the colour maps - 8 bit colour. With 24 bit colour each pixel can tell you

exactly what colour it is, and no amount of fiddling can get it to change it. This is because each pixel, instead of now having an 8 bit number, 0 to 255, which is an index in a colour map, it stores a 24 bit value, 0 - 16.7 million (recognise that number), which tells the pixel exactly what colour to be. The number is a long and contains 8 bits of red, 8 bits of green and 8 bits of blue. This type of colour is called decomposed colour.

Now as far as I can see there are two options of how you can deal with this. Decomposed colour can be switched on or off using device, decomposed = 0 or device, decomposed = 1. So, you can make a 24 bit system still use colour maps. If you put device, decomposed=0 at the start of your program then I'm pretty sure that everything would then work as normal. The advantage though now is that although you are using colour maps, you are not using them in quite the same way as on an 8 bit system. With the 8 bit system, as I said, when you load in a new colour table what's on the screen changes. However if you are using 24 bit colour but with a colour map, the map is merely a translator and what is stored for the pixels is still unchangeable. This is of course a great advantage because you can have different colour tables for each window. This would mean that all the old programs would work but at the same time you would be reaping the rewards of a 24 bit system.

The second option is to go decomposed, and specify all your colours using these 24 bit longs to access the colours. e.g. plot, findgen(100), color=65000 which gives a yellowey colour, no matter what machine you're on and there are no colour table dependencies. David Fanning has a program which converts an RGB triple to the 24 bit long - on the web site I gave at the start. So you could take your arrays and pass them to this program and your st_color array would contain 24 bit longs which were the right colours. I personally think that if your just using colours on plots, or in fact anything except a great deal of image processing then decomposed colour is not really worth, although I'm sure that a programmer with a lot more experience than me will tell you that I'm wrong - so don't take my advice as gospel.

I hope I've explained how the 24 bit colours work, it's a bit hard to get your head round. If what I've said solves the problem, then please post another reply, just to let me know, because I only just a beginner and it helps if what I think is right is confirmed to be right (or quite probably wrong!!)

As a postscript, if there's anyone out there who does animations please e-mail me or post a message to let me know if you would be interested in a replacement for xinteranimate which vastly reduces on pixmap memory usage and enables custom animations to be built very easily. In fact I would boast that there isn't any animation that you couldn't create in under half an hour although for most cases it would be less. You could even have animations where one image was moving every frame and then the other was moving every 3 frames, and for the second animation you would only store

pixmaps for when it was moving, and also any black space in your animations are not stored either.

cheers,
Phil Aldis
