Subject: Re: Object graphic 3d Scatterplot
Posted by Struan Gray on Thu, 23 Mar 2000 08:00:00 GMT
View Forum Message <> Reply to Message

I'm coming to this a bit late, but what the hell....

I played with this a bit when making a suite of routines to draw crystal structures using object graphics. My first approach was the polyline one that David has presented here. The final version simply creates a seperate graphics object for each data point.

The polyline approach is faster, both to create the dataset and to draw it on the screen. Using lots of different individual graphics objects is more versatile and makes some sorts of data picking and user interaction simpler.

Both approaches can be speeded up by limiting the number of symbols. In David's SCATTER_SURFACE each data point has its own symbol. If you can live with the reduced generality, use the fact that a palette for the colours limits the number of distinct symbols to 256. If you re-jig things so that the SYMBOL keyword refers to a 256-entry object array things speed up rather dramatically for 3000 objects. This code *should* drop nicely into David's example if you move the declaration of thisContainer above it:

```
   ; Create a color palette for coloring the symbols.

 thisPalette = Obj_New('IDLgrPalette')
 thisPalette->LoadCT, 5
 thisContainer->add, thisPalette

   ; Create the symbolarray

 npts = n_elements(z)
 nsymbols = min([256, npts])
 theseSymbols=ObjArr(nsymbols)
 FOR i=0,nsymbols-1 DO BEGIN
   if npts le nsymbols then colidx = zcolors[i] else colidx = i
   theseSymbols[i] = Obj_New('IDLgrSymbol', 4, $
      Color=colidx, Size=[0.05, 0.05, 0.05])
 ENDFOR
 if npts gt nsymbols then theseSymbols = theseSymbols[zcolors]
 thisContainer->add, theseSymbols

    ; Create Polyline object..

 thisPolyline = OBJ_NEW('IDLgrPolyline', x, y, z, $
```

LineStyle=6, Symbol=theseSymbols, palette=thispalette)

If you are creating seperate graphics objects you can do the same thing by enclosing the plotted data point in an IDLgrModel and using the /ALIAS keyword to add the symbol objects which you create seperately. To illustrate this, and allow you to play with timings, I have crudely hacked David's program to accept two new keywords and placed it on my web server:

http://www.sljus.lu.se/stm/IDL/misc/scatter_surface.txt

The new keywords are NPTS (the number of data points you want) and POLYTYPE. Polytype=1 (the default) gives you David's solution (though I've fiddled with the colour specification a bit - if you have a palette, why throw it away?). Polytype=2 give you a graphics element for every data point. Polytype=3 minimises the number of graphics elements in 2. Polytype=4 minimises the symbols in David's approach, as above. 2 and 3 are slower than 1 and 4; 3 and 4 are faster than 1 and 2.

Finally, if you only have a few classes of data, and need only a few symbols, you can tidy up the programming by using a different polyline for each class (optionally, with SHARE_DATA). If you are joining up the symbols with lines, using the POLYLINES keyword to specify multiple sub-polylines is also useful. Both of these tricks make it easier to keep track of ownership of the data if you want to change it iteractively.

David Fanning, davidf@dfanning.com writes:

> Andrej Gapelyuk (gapelyuk@fvk-berlin.de) writes:
>
> The only disadvantage which I can see that all symbols are
> flat and non symmetrical in 3d, I would be prefer small
> spheres.
>
> As a matter of fact, it should be possible to use *ANY*
> kind of symbol you like, including 3D symbols (e.g. little
> shaded spheres), since the "value" of the DATA keyword for
> the symbol object can in fact be another model object.

This is true and works nicely. You can pass any IDLgrModel (or simple graphics object) to the IDLgrSymbol's INIT procedure and it will appear at each data point. Using a model lets you use stupidly complex symbols and, more usefully, lets you adjust their colours and other properties on the fly.

Struan
