Subject: Re: Histogram shift
Posted by Michael Wallace on Tue, 22 Feb 2005 06:44:10 GMT
View Forum Message <> Reply to Message

> Well, do tell! We are starved for good news the past week
> or so. :-)

Well, if you really want to know about my world ...

I'd always hated setting hard values for VIEWPLANE_RECT because there was invariably one case that'd throw a wrench into the works and cause your once nice looking plot to look now like crap. For me, it was always my Y axis that'd get pushed off the left side of the viewplane. Given that you're using the same font sizes all the time, you can pretty easily calculate the top and bottom of the viewplane. However, you never really know where the left side is going to end up because the positioning of the text is totally dependent on the length of the tick mark strings on the Y axis. A longer string means your Y title is further out. And if you ever decided that Courier looks better than Helvetica, there you were having to redo the viewplane numbers.

Long story short, I created a function which takes in x and y data vectors as well as title text strings, various fonts, dimensions and a few other inputs and spits out an IDLgrWindow or IDLgrBuffer with the appropriate view, models, IDLgrPlot, etc. stored in the GRAPHICS_TREE. The VIEWPLANE_RECT is calculated based on the positioning of all the graphics objects and so it's never too big or too small. Everything fits just right. I've also now created an equivalent function for an IDLgrImage rather than IDLgrPlot.

And since you're so starved, here's another one but it doesn't deal with Object Graphics. I needed to to use the aforementioned plotting from Java. I didn't want to use the IDL-Java Bridge because the Bridge is clunky and the Bridge only goes one direction. You can instantiate a Java virtual machine in IDL, but you can't instantiate an IDL session from within Java. Anyway, I settled on the easy solution of having my Java process spawn an IDL process. I could form the IDL commands within Java and write the commands to the IDL process and let IDL do it's thing. However, I had to send quite a bit of data across between Java and IDL. Sending the data across the pipe would get very complex very quickly and writing out the data to a file only to be read into IDL could be time consuming. What I wound up doing was using shared memory just as if you wanted to transfer data between two IDL processes. It's quick because you don't hit the I/O subsystems of Java or IDL or of your operating system and it's pretty easy to read and write directly to memory. This solution worked like a champ. I thought it was pretty cool.

Now to wrap things up and put a nice bow on it all, I have a website where I'm using Java to handle the back-end processing and dynamic content.  A user will go to the website and request certain data.  A Java program will then go and get the appropriate data from an Oracle database and send the data on to IDL using the process above.  IDL will then create a plot or two (now with automatic viewplane sizing!) and these images will be served back to the user.  All the user knows is that they click a button and they're taken to a page with their plots on it.  And in a typical case, it only takes a couple seconds for ALL of this to happen.  Naturally, complex plots or querying over huge data ranges will make it slow down, but it's not bad for on-the-fly plot creation.

All in all, it's pretty cool stuff.

-Mike