Subject: Scaling atoms & axes in object graphics
Posted by Mark Hadfield on Wed, 16 Aug 2000 06:03:21 GMT
View Forum Message <> Reply to Message

In the last few days I have been reconsidering my approach to building up scientific graphs in object graphics. By "scientific graphs" I mean a wide class of graphs in which data are represented geometrically in association with axes in 1, 2 or 3 dimensions. (This doesn't rule out the possibility that some aspects of the data are represented non-geometrically, e.g. by colour.) I am weighing the pros and cons of two different ways of handling scaling. Perhaps newsgroup readers would like to comment.

Approach A: Use [X,Y,Z]COORD_CONV

This is the approach which I have built into my MGHgrGraph class (http://katipo.niwa.cri.nz/~hadfield/gust/software/idl/) and its more specialised sub-classes

The steps to building a graph here are:

1. Create a view and a model.

2. Create axes representing the required data range(s), fit them into normalised coordinate range(s) that are visible in the view and add them to the model. By "fit them into the normalised coordinate range" I mean scale the axis ALONG ITS DIRECTION using the appropriate COORD_CONV property: for X axes we adjust XCOORD_CONV, for Y axes we adjust YCOORD_CONV and for Z axes we adjust ZCOORD_CONV.

3. Generate graphics atoms representing the data, add them to the same model and have each atom take its XCOORD_CONV, YCOORD_CONV and ZCOORD_CONV properties from the X, Y & Z axes respectively.

4. Display

Many variations are possible. In particular one might want to calculate the data range after the graphics atoms have been calculated, in which the axes and the atoms all need to be re-scaled.

Approach B: IDLgrModel::Scale & Translate, aka "[X,Y,Z]COORD_CONV is evil"

1. Create a view and a model.

2. Create axes representing the required data range(s) and add them to the model. DON'T adjust their [X,Y,Z]COORD_CONV properties.

3. Generate graphics atoms and add them to the same model. DON'T adjust their [X,Y,Z]COORD_CONV properties.

4. Scale and translate the model as necessary.

5. Display

Pros & Cons:

There is a major advantage to approach B:

Once a set of atoms and axes has been put into a model, any further changes to the scaling of that model will not disturb the geometric relationship between the axes and atoms. With approach A, if you want to fiddle with the scaling of a graph after it has been set up, you have to EITHER: a) keep a list of all the atoms & axes at the top level of the graphics application and apply changes to the [X,Y,Z]COORD_CONV properties of each one; OR b) have each axis keep track of all the atoms that are scaled to it and pass on any changes--then the graphics application just needs to keep track of the axes. I have been experimenting with the latter method via a class I call MGHgrMSaxis ("MS" = master-slave) and it's not as complicated as it sounds, but it's still a level of complication I'd like to avoid.

Advantages of approach A:

Believe it or not, sometimes I like to plot atoms against the same X axis but different Y axes. (This is most often needed in preparing graphs for publication.) It's not impossible to do this with approach A, but it's harder--each combination of horizontal & vertical scaling needs a different model. (Or you scaling all the atoms identically and get one of the Y axes to lie about its scaling via TICKFORMAT, just as you're forced to do in direct graphics--Yecch!)

In approach A, axes can be scaled *only* along their parallel direction, then a property that expresses a position or length in the perpendicular directions, like TICKLEN or LOCATION*, remains in normalised coordinates. So a sensible default for TICKLEN, like 0.05 or so, will work pretty much all the time. With approach B axes have to be scaled in the same directions as their data, which complicates things.

(*) Question for IDL expert programmers, why do I say that LOCATION represents a position perpendicular to the axis, when it has 3 components?

This is an abstruse one: it is possible to have axes that run backwards but look normal by fiddling their TEXTBASELINE and TEXTUPDIR properties. If axes are scaled only in one direction there are only two possibilities for each axis, normal or reverse orientation. With approach B there are 4 (for a two-dimensional graph) or 8 (for a three-dimensional graph), e.g. normal-X reverse-Y, reverse-X normal-Y and so on. This makes my head hurt.

Summary:

The sole advantage I have cited for approach B is a biggy.

What do others think? Is there an approach C I haven't thought of?

Thanks to Randall Frank for setting off this train of thought.

Mark Hadfield
m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield/
National Institute for Water and Atmospheric Research
PO Box 14-901, Wellington, New Zealand