Subject: Re: Working In 3-Space
Posted by Rick Towler on Fri, 13 Feb 2004 01:13:59 GMT
View Forum Message <> Reply to Message

"David Fanning" wrote in message...
> Mike Maroney writes:
>
>> I have a linked list of arrays, each containing a bunch of structures
>> with 3-space verticies P0:(x0,y0,z0)->P1:(x1,y1,z1) of dx length, as
>> well as color info on the segment. What I want to do is plot all the
>> data on one plot, connecting each P0 to each P1 by a tube.
>>
>> I am able to use PLOTS to plot this data with lines, but as soon as 3D
>> shapes enter the picture everything gets screwy. I can use MESH_OBJ
>> to create a cylinder, but I can't seem to orient it or scale it
>> properly; I'm messing around with T3D, but it seems like there must be
>> an easier way to do this.
>
> Uh, I guess. Object graphics.
>
> Rick Towler will be getting back to you ASAP. :-)

Hummm, it seems my beeper is malfunctioning. Sorry for the delay :)

As David said, object graphics is the way to go.

Before you get too fancy, think about how you are going to view this. If your data range is large and you view the whole scene, your tubes are going to look like lines so it may not be worth the hassle. In this case you would be able to use the IDLgrPolyline object similar to using PLOTS with direct graphics.

But if tubes are what you need, then tubes are what you'll get. There are a couple of ways to approach this.

You could use MESH_OBJ to create tubes with local (data) coordinates that are in world coordinates. To do this create a "revolution" object where array1 contains 2 points defining a vector parallel to your V0->V1 vector which is translated along a vector perpendicular to your V0->V1 the distance of the tube radius. P2 would be [x0,y0,z0] and P3 would be [x1,y1,z1]-[x0,y0,z0]. The resulting vertices and polygon array could be passed to IDLgrPolygon, dropped into an instance of IDLgrModel and displayed. No transformation of the objects would be required since the tube will already be oriented and scaled.

The second approach would be to create a "unit length" tube (using mesh object), then transform it to the appropriate place in world coordinates. This approach isn't necessarily better unless repeated calls to MESH_OBJ was

a bottleneck. Which I don't think it will be. If you want to go this route
I can map it out.

Here is an example of the first method. Note that you will have to handle
the special case where (v1-v0) = [0,0,1].

Have fun!

-Rick

```
function makeTube, v0, v1, tubeRadius, p1

    ;  MESH_OBJ params
    p2 = v0
    p3 = v1 - v0

    ;  Calculate the x-product
    ;  You'll have to handle special case where p3==[0,0,1]
    cp = CROSSP(p3,[0.,0.,1.])

    ;  Normalize x-product
    s = SQRT(TOTAL(cp^2))
    cp = cp / s

    ;  Create the line to rotate
    array1 = [[v0],[v1]] - REBIN(tubeRadius * cp, 3, 2)

    ;  Create the mesh
    MESH_OBJ, 6, verts, polys, array1, P1=p1, P2=p2, P3=p3

    ;  Create a polygon and a line representing the original vector
    oTube = OBJ_NEW('IDLgrPolygon', verts, POLYGONS=polys, $
       COLOR=[255,100,50], STYLE=1)
    oVec = OBJ_NEW('IDLgrPolyline', [[v0],[v1]], COLOR=[50,100,255])

    RETURN, [oTube, oVec]

end

pro tubularDude

    tubeRadius = 0.1
    p1 = 20

    oModel = OBJ_NEW('IDLgrModel')

    v0 = [-2.,0.,1.]
```

```
    v1 = [-1.,1.,0.]
    oModel -> Add, makeTube(v0, v1, tubeRadius, p1)

    v0 = [-1.,1.,0.]
    v1 = [0.,0.,0.]
    oModel -> Add, makeTube(v0, v1, tubeRadius, p1)

    v0 = [0.,0.,0.]
    v1 = [1.,1.,1.]
    oModel -> Add, makeTube(v0, v1, tubeRadius, p1)

    XOBJVIEW, oModel, /BLOCK

    OBJ_DESTROY, oModel

end
```