Subject: Re: Displaying Cartesian coordinate data on a sphere
Posted by David Fanning on Tue, 11 Dec 2012 14:17:43 GMT
View Forum Message <> Reply to Message

David Platten writes:

> I have an array of several hundred thousand data points, each with an (x,y,z) coordinate. The coordinates of every point lies somewhere on the surface of a sphere. Each point represents the position and energy of an x-ray photon that has left my Monte Carlo simulation geometry.
>
> I would like to be able to view the data as a texture map overlaid onto a spherical object. I have managed to do this, but I think that the resulting mapping is distorted. I'm not sure how I should configure GRIDDATA so that the resulting grid can be mapped onto the sphere. Should I be using a different command to make the grid? Any help that you can offer would be greatly appreciated. x, y, z and energy are vectors containing the coordinates and energy of the photons:

I'm going to guess that you see some clumping in your output.
Have you read this article:

http://www.idlcoyote.com/math_tips/randomsurface.html

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thue. ("Perhaps thou speakest truth.")

Subject: Re: Displaying Cartesian coordinate data on a sphere
Posted by dplatten on Tue, 11 Dec 2012 15:28:56 GMT
View Forum Message <> Reply to Message

Hi David,

Thanks for the reply. I don't think that it was clumped data - I think it was just my lack of understanding of how the GRIDDATA command works. I've put together a simple test scenario so that I can check my sanity. It behaves as I expect, so I think things are OK. My test creates a data point at the north and south poles, and four around the equator and then interpolates between.

x = [0, 0, 1, -1, 0, 0]
y = [0, 0, 0, 0, 1, -1]
z = [1, -1, 0, 0, 0, 0]
values = [255, 255, 50, 50, 150, 150]

```
grid = GRIDDATA(x, y, z, values, DIMENSION=[30, 30], /SPHERE)

image = BYTSCL(grid)
MESH_OBJ, 4, vertices, polygons, REPLICATE(0.25, 101, 101)
oModel = OBJ_NEW('IDLgrModel')
oPalette = OBJ_NEW('IDLgrPalette')
oPalette -> LOADCT, 33
oPalette -> SetRGB, 255, 255, 255, 255
oImage = OBJ_NEW('IDLgrImage', image, PALETTE = oPalette)
vector = FINDGEN(101)/100.
texure_coordinates = FLTARR(2, 101, 101)
texure_coordinates[0, *, *] = vector # REPLICATE(1., 101)
texure_coordinates[1, *, *] = REPLICATE(1., 101) # vector
oPolygons = OBJ_NEW('IDLgrPolygon',  $
   DATA = vertices, POLYGONS = polygons, $
   COLOR = [255, 255, 255], $
   TEXTURE_COORD = texure_coordinates, $
   TEXTURE_MAP = oImage, /TEXTURE_INTERP)
oModel -> ADD, oPolygons
oModel -> ROTATE, [1, 0, 0], -90
oModel -> ROTATE, [0, 1, 0], -90
XOBJVIEW, oModel
```

Regards,

David

Subject: Re: Displaying Cartesian coordinate data on a sphere
Posted by dplatten on Wed, 12 Dec 2012 09:53:28 GMT
View Forum Message <> Reply to Message

I have found the following solution to be more reliable. It uses the /GRID, XOUT and YOUT switches to ensure that the resulting grid covers the whole sphere. I've realised that the coordinates of the resulting grid are in terms of longitude and latitude. To ensure that I finished up with a grid that covers the entire surface of a sphere I set XOUT to cover a range of +/- PI radians, and YOUT to cover +/- PI/2 radians.

```
x = [0,  0,  1, -1,  0,  0]
y = [0,  0,  0,  0,  1, -1]
z = [1, -1,  0,  0,  0,  0]
values = [255, 0, 100, 50, 0, 150]
maxOut = 2.0*!pi
xout = FINDGEN(31) * 2.0*!pi / 30.0 - (2.0*!pi / 2.0)
yout = FINDGEN(31) * !pi / 30.0 - (!pi / 2.0)
grid = GRIDDATA(x, y, z, values, /GRID, XOUT=xout, YOUT=yout, /SPHERE)
```

```
cgLoadCT, 33
cgWindow, 'cgImage', grid, /KEEP_ASPECT

image = BYTSCL(grid)
MESH_OBJ, 4, vertices, polygons, REPLICATE(0.25, 101, 101)
oModel = OBJ_NEW('IDLgrModel')
oPalette = OBJ_NEW('IDLgrPalette')
oPalette -> LOADCT, 33
oPalette -> SetRGB, 255, 255, 255, 255
oImage = OBJ_NEW('IDLgrImage', image, PALETTE = oPalette)
vector = FINDGEN(101)/100.
texure_coordinates = FLTARR(2, 101, 101)
texure_coordinates[0, *, *] = vector # REPLICATE(1., 101)
texure_coordinates[1, *, *] = REPLICATE(1., 101) # vector
oPolygons = OBJ_NEW('IDLgrPolygon',  $
   DATA = vertices, POLYGONS = polygons, $
   COLOR = [255, 255, 255], $
   TEXTURE_COORD = texure_coordinates, $
   TEXTURE_MAP = oImage, /TEXTURE_INTERP)
oModel -> ADD, oPolygons
oModel -> ROTATE, [1, 0, 0], -90
oModel -> ROTATE, [0, 1, 0], -90
XOBJVIEW, oModel
```

Regards,

David
