Subject: Re: MESH_DECIMATE anyone?
Posted by Karl Schultz on Fri, 03 Nov 2000 08:00:00 GMT
View Forum Message <> Reply to Message

"Andrew" <noymer@my-deja.com> wrote in message
news:8tu10c$tb5$1@nnrp1.deja.com...
> Dear C.l.i-p,
>
> I have a problem, and I *think* that MESH_DECIMATE may be
> my solution, but I'm a little confused. This function was
> introduced in IDL 5.3, which is also the version I am using.

MESH_DECIMATE is intended to simplify polygonal meshes by merging small
polygons into larger ones. Rather than using simple resampling or naive
combining, MESH_DECIMATE carefully picks the polygons it merges so that the
new simplfied mesh is as close to the original as possible. It tends to
combine polygons in "flat" areas and keep polygons in places where the mesh
is rapidly changing.

> Here is the deal: I have a huge matrix of simulated data
> in three dimensions. It's big enough that 3D wireframes are
> too dense (like solid black), and need to be thinned out. One
> solution is to sample the data. This has several appealing
> aspects. The data in this case are regularly-gridded so the
> mechanics of sampling are simple. And since I am simulating
> the data anyway, I could actually just program the thing to output
> less data but to output the correct average of n points, which is
> even better than just sampling.

Yes, but decimation should do a better job.

> But before I do that, I thought I would take MESH_DECIMATE
> out for a spin. But I have become confused (what else is new?)
> if this is really the function I want. My data are a rectangular
> array of z-values, which I plot using the SURFACE procedure (DG).
> The data start of as a list of (x,y,z) triplets, R*C long. Following
> advice I got about a year ago from this newsgroup, I convert the
> data into three separate R*C arrays using the REFORM function.
> The z array is the actual data, and the x and y arrays make sure
> the axes are labeled correctly. All I want to do is thin out the
> z array and make a new, less dense, surface out of it.

This sounds like an excellent application for decimation. Height fields or
anything else that you might be tempted to plot with SURFACE might work
well. But read on.

> The syntax of MESH_DECIMATE is:
> Result = MESH_DECIMATE (Verts, Conn, Connout [, /VERTICES]

> [, PERCENT_VERTICES=percent | , PERCENT_POLYGONS=percent])
>
> where:
> Verts=Input array of polygonal vertices [3, n].
> Conn=Input polygonal mesh connectivity array.
> Connout=Output polygonal mesh connectivity array.
>
> Now, Conn is going to be my z-data (???), and Connout will be the
> decimated surface. I'm quite confused about Verts. My vertices
> are just my simulated data points, so for should I feed to Verts
> my original list of (X,Y,Z) triplets??? I'm having trouble
> visualizing this. Even if I can get a nicely-thinned-out z-array,
> what happens to my x and y arrays?

MESH_DECIMATE expects a polygonal mesh for input. A polygonal mesh is, in general, a list of vertices (XYZ triplets) and a connectivity list that describes how these vertices are connected. See the discussions of polygon formats that are used by the IDLgrPolygon object in Object Graphics.

If you start out with a rectangular array of Z values, here is what you have to do:

1) Generate the implicit X and Y coords. Looks like you already know how to do this.
2) Get the X, Y and Z into a (3,n) array. This is the verts argument for input to MESH_DECIMATE.
3) The next step is a bit harder. You have to create a "polygon list". It sounds like you don't have any explicit connectivity info at the start of all this, so you can go ahead and make your own mesh. If the data is regularly gridded, it makes sense to make the mesh out of little squares. For example, if you have a 10x10 surface, then the verts can be stored in the verts array as a (3,100) array. If you keep the vertices ordered carefully in this array, the first polygon in the list would be represented as [4, 0, 1, 11, 10]. This would be the first 5 elements of your conn array. You will end up with 9x9 squares for a total of 9x9x5 entires in your conn array. You could also create triangles [3,0,1,10,3,1,11,10,....], but that's a bit more work and MESH_DECIMATE chops it up into triangles anyway.

So now you have your verts and conn arguments for MESH_DECIMATE. Pick a percentage. For example, 20 means that the decimated mesh will have 20% of the vertices (or polygons) of the original mesh.

MESH_DECIMATE returns a new connectivity list that refers to the vertices in the original vertex list. The new conn array defines polygons that use a subset of the original vertex list. You could use MESH_VALIDATE to get rid of the unused vertices, but sometimes it is convenient to just keep one vertex list and have a few conn lists for each level of detail that you

might want. Or, you may need to keep the original vertex list anyway, so
there is no point in keeping a subset list around.

The /VERTICES keyword (actually a doc error - should be VERTICES=out_verts)
instructs MESH_DECIMATE to move the input vertices around in order to
optimize the mesh and return the new vertex list in out_verts. This allows
the meshes to be slightly better (closer to the original).

>
> Usually I test things on small datasets before I do it on a big
> dataset---but how do I decimate a 3x3 array? :-0

In this case, larger tests are a bit better, to see the effects of
decimation.

I suggest looking at the decimate.pro example shipped with IDL in the
examples directory. Just type "decimate" at the IDL command line to run the
example. In IDL 5.3, there is a cow model that you can interactively
decimate. In 5.4 we changed this model to a height field (Maroon Bells -
mountains). In both cases, you can play with the decimation percentage and
watch how the mesh changes. In the 5.4 version, there is code that reads Z
data and makes the polygon list, as I described above:

```
 dim = 64
 heightField = BYTARR(dim*dim, /NOZERO)
 OPENR, lun, /GET_LUN, filename, ERROR=err
 IF (err NE 0) THEN BEGIN
  PRINTF, -2, !ERR_STRING
  RETURN
 ENDIF
 READU, lun, heightField
 CLOSE, lun
 FREE_LUN, lun

 ; Generate 2D grid for X and Y.
 ; Fill in Z with height data.
 coords = (FINDGEN(dim)/(dim-1)) # REPLICATE(1, dim)
 verts = FLTARR(3, dim*dim)
 verts[0,*] = REFORM(coords, dim*dim)
 verts[1,*] = REFORM(TRANSPOSE(coords), dim*dim)
 verts[2,*] = FLOAT(heightField) / (1.5 * MAX(heightField))

 ; generate quad mesh connectivity list
 i = 0L
 faces = LONARR(5 * (dim-1) * (dim-1))
 FOR y = 0, dim-2 DO BEGIN
  FOR x = 0, dim-2 DO BEGIN
   basevert = y * dim + x
   faces[i] = [4,basevert, basevert+1, basevert+dim+1, basevert+dim]
   i = i + 5
  ENDFOR
 ENDFOR
```

The rest of decimate.pro should also give you some more ideas on using MESH_DECIMATE.

> Beore I just go back to sampling the data, I'd be grateful if
> someone out there who has become an expert in MESH_DECIMATE could
> point me in the right direction. For regularly gridded data, I'm
> beginning to think that MESH_DECIMATE may be overkill---but I'd be
> interested to learn if this is incorrect.

Hope this helps. I have a white paper about polygonal meshes in IDL that I'll send to you if you can let me know where to send it.

Karl
RSI