Subject: Re: simplify a polyline?
Posted by Karl Schultz on Mon, 26 Apr 2004 17:44:28 GMT
View Forum Message <> Reply to Message

"Ben Tupper" <btupper@bigelow.org> wrote in message
news:c6j2sf$coqvb$1@ID-189398.news.uni-berlin.de...
> Brad Gom wrote:
>
>> I need a general purpose routine for reducing the complexity of a 2-d
>> polyline. For example, the output of the contour function contains
>> many redundant points, ie. many vertices may be removed as they fall
>> on a straight or nearly straight line. Has anyone implemented a
>> polyline simplification or decimation routine? I don't want to simply
>> smooth the input data.
> Hi,
>
> I haven't tried this, but I believe that you can use MESH_DECIMATE using
> your X, Y vertices coupled with a faked Z value. This is from the
> online description of MESH_DECIMATE
>
>> The MESH_DECIMATE function reduces the density of geometry
>> while preserving as much of the original data as possible.

Here is a program that does what Ben suggests.

Karl

```
PRO coastline

   filename = FILEPATH('states2.sav',
SUBDIRECTORY=['examples','demo','demodata'])
   RESTORE, filename

   ; pick a state and get its outline data
   n = 57
   PRINT, "State is ",  states[n].state
   outline = *states[n].poutline
   ; free stuff we do not need
   PTR_FREE, states.poutline

   ; build vertex array of outline, plus another copy of the outline
stacked on top in Z
   nPoints = N_ELEMENTS(outline[0,*])
   pts = FLTARR(3,nPoints*2)
   pts[0,0:nPoints-1] = outline[0,0:nPoints-1]
   pts[1,0:nPoints-1] = outline[1,0:nPoints-1]
   pts[2,0:nPoints-1] = 0
   pts[0,nPoints:2*nPoints-1] = outline[0,0:nPoints-1]
```

```
   pts[1,nPoints:2*nPoints-1] = outline[1,0:nPoints-1]
   pts[2,nPoints:2*nPoints-1] = 10

   ; build connectivity array to make quads between the two outlines.
   ; this will look like an extrusion of the outline
   conn = LONARR(5 * nPoints)
   conn[LINDGEN(nPoints)*5] = 4
   conn[LINDGEN(nPoints)*5+1] = LINDGEN(nPoints)
   conn[LINDGEN(nPoints)*5+2] = LINDGEN(nPoints) + 1
   conn[LINDGEN(nPoints)*5+3] = LINDGEN(nPoints) + nPoints + 1
   conn[LINDGEN(nPoints)*5+4] = LINDGEN(nPoints) + nPoints
   conn[5 * nPoints - 3] = nPoints
   conn[5 * nPoints - 2] = 0

   ; look at the original extrusion
   oPolygon1 = OBJ_NEW('IDLgrPolygon', pts, POLYGON=conn, COLOR=[255,0,0])
   xobjview, oPolygon1

   ; decimate and look at the decimated extrusion
   n = MESH_DECIMATE(pts, conn, new_conn, PERCENT_VERTICES=50)
   oPolygon2 = OBJ_NEW('IDLgrPolygon', pts, POLYGON=new_conn,
COLOR=[0,255,0])
   xobjview, oPolygon2

   ; Now pull out the vertices that remain after the decimation

   ; First, filter out the 3's from the conn list
   ; Replace the 3's with a "big" value that we'll filter out later
   i = LINDGEN(N_ELEMENTS(new_conn)/4)*4
   line_conn = new_conn
   line_conn[i] = nPoints

   ; Now keep only the vert indicies from the decimated list that are
   ; smaller than nPoints.  This gets rid of all the verts from the top
   ; of the extruded outline.
   i = WHERE(line_conn LT nPoints)
   line_conn = line_conn[i]

   ; Now sort and uniq the list, so that we only get one of each vertex,
   ; and in the right order.
   ; Otherwise, we'd have duplicate verts introduced by the triangles.
   line_conn = line_conn[UNIQ(line_conn, SORT(line_conn))]

   PRINT, nPoints, ' points in the original (red) outline.'
   PRINT, N_ELEMENTS(line_conn), ' points in the decimated (green)
outline.'

   oPolyline1 = OBJ_NEW('IDLgrPolyline', pts[*, 0:nPoints-1],
```

```
COLOR=[255,0,0])
   oPolyline2 = OBJ_NEW('IDLgrPolyline', pts[*,line_conn], COLOR=[0,255,0])
   xobjview, [oPolyline1, oPolyline2], /BLOCK
   OBJ_DESTROY, [oPolygon1, oPolygon2, oPolyline1, oPolyline2]
END
```