Subject: Re: 3D point cloud visualization: filled polygons in the front, different fill colour + lines in the back
Posted by Nuno Ferreira on Thu, 28 May 2015 20:08:36 GMT
View Forum Message <> Reply to Message

Thanks Dick, it helped me a lot. Using Compute_Mesh_Normals() is a nice solution that worked well (after some testing... it would also be great if we could use many offset values with the DEPTH_OFFSET keyword, using the exact vertices positions, but apparently that is not the case).

I have managed to show the 3D point cloud as filled color polygons in the front side and filled grey polygons in the back, with layers of points and/or lines in each side (front and back), using different colors and transparency levels (I am using black for the back face and white for the front face but it could be any color, of course). Here is a screenshot showing an example of the two sides of the surface: https://drive.google.com/open?id=0B6Ti5FMqve-dMHBTXzJpSHQ1Q2M&authuser=0 (I agree it is not the best test object... :)).

Here is the code I am using, in case it may help others. It probably has unnecessary statements such as DOUBLE, DEPTH_TEST_DISABLE, etc that were added during the tests. I am using slider widgets to set the transparency of each layer independently (via the "vis_alpha_*" parameters below):

```
offset_factor = 0.05       ; defines the distance between the
                            ; different layers.

;  layer 0: filled polygons (color in the front, grey in the back):
p  = idlgrpolygon(v, poly=c, vert_colors=vc, $
  style=vis_style, shading=vis_shading, $
  bottom=[200,200,200], depth_offset=0, /double)
normalsXYZ = Compute_Mesh_Normals(v, c)

;   layer +1: lines with the same colors as the filled polygons
;   (the idea was to use this to help covering some points from
;   the back that sometimes appear in the front face, when I zoom out.
;   It didn't work - instead it is being used to give some color
;   to the back face, if needed...)
vc2 = vc
vc2[3,*] = vis_alpha_color_lines*255
v2 = v + normalsXYZ * offset_factor
p2 = idlgrpolygon(v2, poly=c, vert_colors=vc2, $
  style=1, shading=vis_shading, depth_offset=0, $
  depth_test_function=4, depth_test_disable=2, /double)

;  layer -1: lines in the back
v3 = v - normalsXYZ * offset_factor
p3 = OBJ_NEW('IDLgrPolygon', v3, POLYGONS=c, $
  STYLE=1, color=[0,0,0], depth_offset=1, $
  alpha=vis_alpha_lines_back, depth_test_function=2, $
```

```
  depth_test_disable=2, /double)

;   layer -2: points in the back
v4 = v - normalsXYZ * offset_factor * 2
p4 = OBJ_NEW('IDLgrPolygon', v4, POLYGONS=c, STYLE=0, $
  color=[0,0,0], depth_offset=1, $
  alpha=vis_alpha_points_back, depth_test_function=2, $
  depth_test_disable=2, /double)

;   layer +2: lines in the front
v5 = v + normalsXYZ * offset_factor * 2
p5 = OBJ_NEW('IDLgrPolygon', v5, POLYGONS=c, STYLE=1, $
  color=[255,255,255], depth_offset=0, $
  alpha=vis_alpha_lines_front, depth_test_function=2, $
  depth_test_disable=2, /double)

;   layer +3: points in the front
v6 = v + normalsXYZ * offset_factor * 3
p6 = OBJ_NEW('IDLgrPolygon', v6, POLYGONS=c, STYLE=0, $
  color=[255,255,255], depth_offset=0, $
  alpha=vis_alpha_points_front, depth_test_function=2, $
  depth_test_disable=2, /double)
```

It is probably overkill, but it is nice to have full control of what we see...

Nuno