Subject: 3D point cloud visualization: filled polygons in the front, different fill colour + lines in the back
Posted by Nuno Ferreira on Mon, 25 May 2015 17:56:29 GMT
View Forum Message <> Reply to Message

Hi all,

I am trying to visualize 3D point clouds taken with a Kinect v2 camera. My data consist of a set of vertices in 3D space (xyz + RGBA) with connectivity information (polygons). Suppose I take a photo of someone, with depth information, from a single position: looking from the front, I would see a normal photo, but it could be rotated all around, allowing to see the back (I would have the polygons only from the half-surface that was facing the camera). I am trying to visualize this dataset using filled polygons in the front side (without lines) and shades of grey + lines in the back face.

I tried a few options but each has specific problems (please see the screenshot: https://drive.google.com/file/d/0B6Ti5FMqve-dRzRuMk9yY1FjM2c /view?usp=sharing - you can also run the program below):

- in option 1, it is difficult to distinguish the back face from the front face and that is why I would prefer using a single color for the back face (e.g., grey, eventually shaded + lines);

- in options 1 and 2, the lines in the front side cover the filled polygons and since the density of vertices is large I would see mainly black lines that would at least partially hide the colors of the polygons;

- option 3 is almost what I would like, but I would prefer having the lines in the back face with a single color. Do you have any suggestions?

I am also aware that I could either offset or scale the vertices positions to avoid covering the lines with the polygons but I would prefer a solution that does not change these positions.
Many thanks and sorry for the long message!

Nuno

```
PRO question_example_event, ev
; (not really needed for this example)
END


PRO question_example
; create a test object
vertices = [[0, 0, 0], [1, 0, 0], [1, 1, 0], [0, 1, 0], [0.5, 0.5, 1]]
connectivity = [3, 0, 1, 4, $
                3, 1, 2, 4, $
```

```
            3, 2, 3, 4, $
            3, 3, 0, 4]
vert_colors = [[0,0,0],[255,0,0],[0,255,0],[0,0,255],[255,255,255]]

; Initialize model for display (2 views).
oModel1 = OBJ_NEW('IDLgrModel')
oModel2 = OBJ_NEW('IDLgrModel')

; Initialize polygon and/or polyline
option=4
CASE option of

  1:    begin
         oPolygon  = OBJ_NEW('IDLgrPolygon', vertices, $
           POLYGONS = connectivity, SHADING=1, $
           vert_colors=vert_colors)
         oPolyline = OBJ_NEW('IDLgrPolyline', vertices, $
           POLYLINES = connectivity, COLOR = [0, 0, 0])
        end

   2:   begin
         oPolygon  = OBJ_NEW('IDLgrPolygon', vertices, $
           POLYGONS = connectivity, SHADING=1, $
           vert_colors=vert_colors, bottom=[200,200,200], $
           depth_offset=1)
         oPolyline = OBJ_NEW('IDLgrPolyline', vertices, $
           POLYLINES = connectivity, COLOR = [0, 0, 0])
        end

  3:    begin
         oPolygon  = OBJ_NEW('IDLgrPolygon', vertices, $
           POLYGONS = connectivity, SHADING=1, $
           vert_colors=vert_colors, bottom=[200,200,200], $
           depth_offset=1)
         oPolygon2  = OBJ_NEW('IDLgrPolygon', vertices, $
           POLYGONS = connectivity, SHADING=1, STYLE=2, $
           vert_colors=vert_colors, bottom=[200,200,200], $
           depth_offset=0)
        end
endcase

; Add the polygon(s) and/or polyline to the model.
oModel1 -> Add, oPolygon
if option EQ 1 or option EQ 2 then oModel1 -> Add, oPolyline
if option EQ 3 then oModel1 -> Add, oPolygon2
oModel2.Add, oModel1, /alias   ; create an alias for 2nd model

; used for display:
```

```
ov1 = idlgrview(viewplane_rect=[-2, -2, 4, 4], zclip=[2, -2], eye=2)
ov1.add, oModel1
ov2 = idlgrview(viewplane_rect=[-2, -2, 4, 4], zclip=[2, -2], eye=2)
ov2.add, oModel2

; create GUI
s     = 256
base  = widget_base(/row, Title='Option '+string(option))
d1    = widget_draw(base, graphics_level=2, $
  xsize=s, ysize=s, tooltip="Front")
d2    = widget_draw(base, graphics_level=2, $
  xsize=s, ysize=s, tooltip="Back")

; show GUI and model
widget_control, base, /realize
widget_control, d1, get_value=ow1
widget_control, d2, get_value=ow2
oModel1 -> Rotate, [1, 0, 0], 45   ; Rotate to better show the front side
ow1.setproperty, graphics_tree=ov1
ow1.draw
oModel1 -> Rotate, [1, 0, 0], 180  ; Rotate again to show the back side
ow2.setproperty, graphics_tree=ov2
ow2.draw

xmanager, 'question_example', base
END
```