Subject: Re: 3D point cloud visualization
Posted by Jim Pendleton on Wed, 29 Apr 2015 19:14:34 GMT
View Forum Message <> Reply to Message

On Wednesday, April 29, 2015 at 10:18:04 AM UTC-6, Nuno Ferreira wrote:
> Thank you very much for your fast replies, they were most helpful! Since I now have to change my widget application that presents several views of the 3D point cloud, I have a few more questions... (since they relate to the same topic, I keep them in the same thread).
>
>
> 1 - I could use the POLYGONS function ("function graphics") with my widget application by adding a WIDGET_WINDOW. However, I could not do the same with the "object graphics" routine XObjView: it keeps showing the point cloud in a separate window instead of inside the main application window. How can I make XObjView to draw the point cloud inside e.g. a draw widget? (BTW, I read this group's thread "Object Graphics and Widgets" that explains the basics)
>
>
> 2 - With object graphics (XObjView and IDLgrPolygon) I can easily rotate the point cloud in 3D by using the mouse, which is great. However, with "function graphics" POLYGON() I only could move and resize the data, not rotate it. Is there a way to interactively rotate in 3D for this case? (I could not find this in the documentation, sorry)
>
>
> 3 -Finally, if you have suggestions on having multiple 3D views (3, for instance) of the same data inside a widget application, I'm all ears...
>
> Thanks in advance,
> Nuno

I've looked at the Kinect API but have yet to write anything with it, so I'd like to see what you come up with as well. Please share some screen shots if you get a chance and you're allowed.

I think you're requiring enough custom code that I would steer you toward the lower level Object Graphics rather than function graphics. The trackball object will give you the 3D functionality you need. You can add your own models and event handling for translation and scaling. It would be a fair amount of work to try to put XOBJVIEW functionality directly into your existing widget. It was originally designed as a demo and wasn't designed to be embedded, which is unfortunate since it's one of the more useful tools for quickly viewing 3D model trees.

Below is some stripped down example code that could help you initially. Also see the IDL Data Point blog article I wrote back in September 2014 that includes a discussion of aliasing model trees: http://tinyurl.com/ohe6377

```
pro tball_event, e
widget_control, e.top, get_uvalue = tb
if (tb.update(e, transform = txf)) then begin
```

```
    d = widget_info(e.top, /child)
    widget_control, d, get_value = ow
    ow.getproperty, graphics_tree = ov
    om = ov.get()
    om.getproperty, transform = mxf
    om.setproperty, transform = mxf # txf
    ow.draw
endif
; You could add other event handling for zoom and translate here
; since trackball only manages rotation.
; Consider adding nested models to handle rotation, translation
; and scaling separately.
end


pro tb
; Make a cylinder
t = findgen(30)*12*!dtor
x = cos(t)
y = sin(t)
z = replicate(-2.5, t.length)
s = 512
mesh_obj, 5, v, c, transpose([[x],[y],[z]]),p1 = 2, p2 = [0, 0, 5], /closed
; Create a model and add orb objects at each vertex, each with its own color
om = idlgrmodel()
vc = bytarr(3, v.length/3)
for i = 0l, v.length/3 - 1 do begin
    vc[*, i] = [i *3 mod 256, i * 5 mod 256, 255 - i*7 mod 256]
    om.add, orb(pos = v[*, i], density = 1, radius = .1, color = vc[*, i])
endfor
; Add the cylinder itself to the model: optional if you have connectivity
;p = idlgrpolygon(v, poly = c, vert_colors = vc, style = 2, shading = 1)
;om.add, p
; Create a view space to hold the cylinder
ov = idlgrview(viewplane_rect = [-5., -5., 10, 10], zclip = [5, -5], eye = 6.)
ov.add, om
; Consider adding ALIAS models to contain alternative views of the same
; source model, so you don't need to maintain multiple copies.
; Simply modify the transform of the model containing the alias.
; Create a widget draw in a base
w = widget_base()
d = widget_draw(w, graphics_level = 2, xsize = s, ysize = s, $
    /button_events, /motion_events)
; Create a trackball to handle left-button drag events for rotation
tb = trackball([s/2, s/2], s/2)
widget_control, w, set_uvalue = tb
widget_control, w, /realize
widget_control, d, get_value = ow
```

```
ow.setproperty, graphics_tree = ov
ow.draw
xmanager, 'tball', w, /no_block
end
```