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
```

Subject: Re: 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 18:01:02 GMT
View Forum Message <> Reply to Message

> ; Initialize polygon and/or polyline
> option=4

I am sorry, "option" should be 1, 2 or 3.

Subject: Re: 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 18:10:47 GMT
View Forum Message <> Reply to Message

And there was another error, sorry (wrong version...)! Here is the right version:

```
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=3
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, STYLE=2, $
           vert_colors=vert_colors, bottom=[200,200,200], $
           depth_offset=1)
         oPolygon2  = OBJ_NEW('IDLgrPolygon', vertices, $
           POLYGONS = connectivity, SHADING=1, STYLE=1, $
           vert_colors=vert_colors, $
```

```
          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=2.1)
ov1.add, oModel1
ov2 = idlgrview(viewplane_rect=[-2, -2, 4, 4], zclip=[2, -2], eye=2.1)
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
```

Subject: Re: 3D point cloud visualization: filled polygons in the front, different fill colour + lines in the back
Posted by Dick Jackson on Tue, 26 May 2015 01:03:26 GMT
View Forum Message <> Reply to Message

Hi Nuno,

You're nearly there, with option 3... on the oPolygon2, you had VERT_COLORS set, which gave every vertex its own colour, but if you just remove that, the lines will be black. Or, if you want a certain colour to be used (with this STYLE=1 wireframe), COLOR=[r,g,b] works, as does VERT_COLORS with a single [r,g,b] triple. I would go with COLOR as the simpler option.

Nice job!

Cheers,
-Dick

Dick Jackson Software Consulting Inc.
Victoria, BC, Canada --- http://www.d-jackson.com

On Monday, 25 May 2015 11:10:48 UTC-7, Nuno Ferreira wrote:
> And there was another error, sorry (wrong version...)! Here is the right version:
>
>
>
> 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=3
> 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, STYLE=2, $
>            vert_colors=vert_colors, bottom=[200,200,200], $
>            depth_offset=1)
>          oPolygon2  = OBJ_NEW('IDLgrPolygon', vertices, $
>            POLYGONS = connectivity, SHADING=1, STYLE=1, $
>            vert_colors=vert_colors, $
>            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=2.1)
> ov1.add, oModel1
> ov2 = idlgrview(viewplane_rect=[-2, -2, 4, 4], zclip=[2, -2], eye=2.1)
> 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
```

Subject: Re: 3D point cloud visualization: filled polygons in the front, different fill colour + lines in the back
Posted by Nuno Ferreira on Tue, 26 May 2015 08:14:52 GMT
View Forum Message <> Reply to Message

Thanks Dick, but in that case the black lines are drawn also in the front face, hiding the colors of the polygons (and with thousands of very small polygons, one would see mainly black in the front... you can see a sample of my data here, shown with option 3: https://drive.google.com/file/d/0B6Ti5FMqve-da0RXNllHUFJfSlk /view?usp=sharing).

What I would like to have, looking at the other image (https://drive.google.com/file/d/0B6Ti5FMqve-dRzRuMk9yY1FjM2c /view) is this: option 3 in the front and option 2 in the back.

In more general terms, I guess I would like to have the possibility of viewing points, lines and filled polygons with certain settings in the front face, and different settings in the back face.

Many thanks!

Nuno

Subject: Re: 3D point cloud visualization: filled polygons in the front, different fill colour + lines in the back
Posted by Dick Jackson on Tue, 26 May 2015 17:27:50 GMT
View Forum Message <> Reply to Message

On Tuesday, 26 May 2015 01:14:55 UTC-7, Nuno Ferreira wrote:
> Thanks Dick, but in that case the black lines are drawn also in the front face, hiding the colors of the polygons (and with thousands of very small polygons, one would see mainly black in the front... you can see a sample of my data here, shown with option 3: https://drive.google.com/file/d/0B6Ti5FMqve-da0RXNllHUFJfSlk /view?usp=sharing).

Ah, of course. :-)

> What I would like to have, looking at the other image (https://drive.google.com/file/d/0B6Ti5FMqve-dRzRuMk9yY1FjM2c /view) is this: option 3 in the front and option 2 in the back.
>
> In more general terms, I guess I would like to have the possibility of viewing points, lines and filled polygons with certain settings in the front face, and different settings in the back face.

Then I think we have to get tricky... To avoid the wireframe lines showing through to the *front*, we can't use DEPTH_OFFSET in a way that works from every point of view (DEPTH_OFFSET lets you push one object's apparent draw-ordering "away" from you at all times). Here's a way to make a shifted set of vertices for the wireframe that is always "inside" the mesh of the shaded surface:

```
    oPolygon  = OBJ_NEW('IDLgrPolygon', vertices, $ ; Shaded surface
      POLYGONS = connectivity, SHADING=1, STYLE=2, $
      vert_colors=vert_colors, bottom=[200,200,200], $
      depth_offset=0)
    normalsXYZ = Compute_Mesh_Normals(vertices, connectivity)
    vertices2 = vertices - normalsXYZ * 0.02 ; Magic number!
    oPolygon2  = OBJ_NEW('IDLgrPolygon', vertices2, $ ; Wireframe
      POLYGONS = connectivity, STYLE=1, $
      depth_offset=1)
```

That magic number happens to work well here. Try 0.01 (some bits of lines poke through the front) and 0.1 (noticeable gap between wireframe and shaded surface) to reveal the mystery. Your models may require a different value for good performance.

Also, see my "***" notes for a couple of other tips to allow you to keep windows on the screen for easy comparison.

> Many thanks!
>
> Nuno

You're welcome!

-Dick

Dick Jackson Software Consulting Inc.
Victoria, BC, Canada --- http://www.d-jackson.com

;-----

```
PRO nuno_question_example_1_event, ev
  ; (not really needed for this example)
END


PRO nuno_question_example_1
  ; 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=3
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, $ ; Shaded surface
      POLYGONS = connectivity, SHADING=1, STYLE=2, $
      vert_colors=vert_colors, bottom=[200,200,200], $
      depth_offset=0)
    normalsXYZ = Compute_Mesh_Normals(vertices, connectivity)
    vertices2 = vertices - normalsXYZ * 0.02 ; Magic number!
    oPolygon2  = OBJ_NEW('IDLgrPolygon', vertices2, $ ; Wireframe
      POLYGONS = connectivity, STYLE=1, $
      depth_offset=1)
  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=2.1)
ov1.add, oModel1
ov2 = idlgrview(viewplane_rect=[-2, -2, 4, 4], zclip=[2, -2], eye=2.1)
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", retain=2) ;***
  d2    = widget_draw(base, graphics_level=2, xsize=s, ysize=s, $
                tooltip="Back", retain=2) ;***

  ; 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, 'nuno_question_example_1', base, /no_block ;***
END
```

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

Subject: Re: 3D point cloud visualization: filled polygons in the front, different fill colour + lines in the back
Posted by Nuno Ferreira on Fri, 29 May 2015 09:28:30 GMT
View Forum Message <> Reply to Message

> This is a very cool use of the visualization capabilities of IDL. Is there an ultimate goal or are you just enjoying the fun of experimentation with new tools?

Yes, there is a goal. We are assessing the use of 3D sensing devices in medical imaging, namely in image fusion and movement analysis/correction. We would like to use the 3D surface information provided by these sensors as an aid to fuse patient data acquired in different scanners. For instance, we would like to provide CT information to a PET-only scanner (CT and PET stand for Computed Tomography and Positron Emission Tomography respectively; most PET scanners nowadays also have CT, but some prototypes and old cameras only have PET). Another possibility is to measure and correct patient movement that may occur during studies performed in PET/CT scanners (since CT and PET data are not acquired simultaneously, some mis-registration of the two types of information may occur if there is movement). There are also other potential applications, such as using the Kinect to help positioning the patient in the scanner if the camera does not have a patient positioning system.

> Thanks for the screen shots!

You're welcome.

> (As of Friday morning I no longer work for Exelis - I now work for Harris.)

Best luck on your new job!

Nuno

Subject: Re: 3D point cloud visualization: filled polygons in the front, different fill colour + lines in the back
Posted by Jim Pendleton on Fri, 29 May 2015 13:06:21 GMT
View Forum Message <> Reply to Message

On Friday, May 29, 2015 at 3:28:33 AM UTC-6, Nuno Ferreira wrote:
>> This is a very cool use of the visualization capabilities of IDL. Is there an ultimate goal or are you just enjoying the fun of experimentation with new tools?
>
> Yes, there is a goal. We are assessing the use of 3D sensing devices in medical imaging, namely in image fusion and movement analysis/correction. We would like to use the 3D surface information provided by these sensors as an aid to fuse patient data acquired in different scanners. For instance, we would like to provide CT information to a PET-only scanner (CT and PET stand for Computed Tomography and Positron Emission Tomography respectively; most PET scanners nowadays also have CT, but some prototypes and old cameras only have PET).

Another possibility is to measure and correct patient movement that may occur during studies performed in PET/CT scanners (since CT and PET data are not acquired simultaneously, some mis-registration of the two types of information may occur if there is movement). There are also other potential applications, such as using the Kinect to help positioning the patient in the scanner if the camera does not have a patient positioning system.
>
>
>> Thanks for the screen shots!
>
> You're welcome.
>
>
>> (As of Friday morning I no longer work for Exelis - I now work for Harris.)
>
> Best luck on your new job!
>
> Nuno

This sounds like a very interesting project.

My job is (should be?) the same. We simply have new ownership!

Jim P
"I work for Harris"

Jim P.
