
Subject: Re: semi-transparent IDLgrPolygon? alpha blending no good...

Posted by [David Fanning](#) on Tue, 23 Apr 2002 13:10:48 GMT

[View Forum Message](#) <> [Reply to Message](#)

Sean Dettrick (dettrick@uci.edu) writes:

> Does anyone have a way to make semi-transparent polygon
> objects with IDLgrPolygon?
>
> I would like to use IDL's fancy object graphics, but in my
> attempts so far, alpha blending using a monochrome image only
> makes a polygon object transparent to ITSELF, not to other
> objects. This is pretty useless.
>
> I would like the red cylinder to be semi-transparent, so
> that I can see the black cylinder through it.

I think the problem is the order in which you put
your polygons in the model. I futzed around with
a couple of things that didn't work, then I took
the polygon you call "thisSurface" and added it
to the model *after* the other polygons and Viola!,
just what I wanted!

I'm not sure now exactly what I changed, but here
are the relevant bits, I think.

; Changed the order of the polygons in model

```
if keyword_set(light) then thisModel->Add, light
if keyword_set(surface2) then thisModel->Add, surface2
if keyword_set(surface3) then thisModel->Add, surface3
if keyword_set(surface4) then thisModel->Add, surface4
thisModel->Add, thisSurface
```

; Here is how I built the drape for the image object.

```
alpha = replicate(128,2) # replicate(1,2)
red = replicate(200,2) # replicate(1,2)
green = replicate(20,2) # replicate(1,2)
blue = replicate(20,2) # replicate(1,2)
drape = Bytarr(4, 2, 2)
drape = [[[red]],[[green]],[[blue]],[[alpha]]]
```

; I made the color of drape image red, so I made the
; polygon color white.

```
thisSurface = OBJ_NEW('IDLgrPolygon', x, y, z, Style=2, Shading=1, $
```

```
    polygon=polygon, color=[255,255,255], $  
    texture_coord= texture_coord,$  
    texture_map=image )
```

If you can't get that to work, let me know and I'll send you the code I have.

Cheers,

David

--

David W. Fanning, Ph.D.

Fanning Software Consulting

Phone: 970-221-0438, E-mail: david@dfanning.com

Coyote's Guide to IDL Programming: <http://www.dfanning.com/>

Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: semi-transparent IDLgrPolygon? alpha blending no good...

Posted by [Rick Towler](#) on Tue, 23 Apr 2002 17:06:57 GMT

[View Forum Message](#) <> [Reply to Message](#)

As David pointed out, you need to be aware of the order in which atoms are added to your model when rendering scenes with alpha blending.

To put it simply, when IDL renders a transparent atom it calculates the color of the atom's surface as a function of it's own color, your alpha value, and the color of objects behind it. Much to my disappointment, atoms in a scene aren't rendered magically all at once but are rendered in the order that they are added to your model hierarchy. So in your case, if a transparent atom is rendered before anything else in the scene it appears "transparent to ITSELF". Adding the transparent cylinder after all of the other atoms as David did corrects the problem by rendering the atoms that appear behind the transparent cylinder first.

Of further note, the order that the transparent atom is meshed will affect what you see as well. In your example program (with the correct ordering of atoms in the model) the initial orientation of the model results in the expected image. You see the back surface of the cylinder thru the front. If you rotate the object 180 degrees about the Y axis, you will note that you no longer see the back surface thru the front. This is due to the fact that your cylinder is meshed from -Z to +Z and with the initial view when the front of the cylinder is drawn, the back surface exists and the color of the front surface reflects this. But, when you view your cylinder from behind, the front surface is drawn before the back and you no longer see the back thru the front.

This may not be an issue for you, but the solution is to break up your transparent atom into a -Z to 0 and 0 to +Z pair and then manage their order in your model (use `mesh_clip` and `IDLgrModel`'s `Move` method). As long as you know where the viewer's eye is, you can make sure the back is drawn before the front.

-Rick

"Sean Dettrick" <dettrick@uci.edu> wrote in message
news:3CC5191F.F0951184@uci.edu...

>
> (apologies for the attachment)
>
> Hi,
>
> Does anyone have a way to make semi-transparent polygon
> objects with `IDLgrPolygon`?
>
> I would like to use IDL's fancy object graphics, but in my
> attempts so far, alpha blending using a monochrome image only
> makes a polygon object transparent to ITSELF, not to other
> objects. This is pretty useless.
>
> Here is my example code, where I have rearranged
> David Fanning's `object_shade_surface.pro` (thanks DF)
> to instead plot up four cylindrical `IDLgrPolygon` objects.
> You can run it with:
> IDL> .r cylinder3
>
> I would like the red cylinder to be semi-transparent, so
> that I can see the black cylinder through it.
>
> I am away of Struan Gray's direct graphics tutorial,
> and guess one could do something similar with defining
> a z-buffer in the view object, but it is way above my head.
>
> Any assistance/pointers greatly appreciated.
>
> Thanks v much,
>
> Sean Dettrick,
> Physics Dept,
> University of California, Irvine

Subject: Re: semi-transparent IDLgrPolygon? alpha blending no good...

Posted by [David Fanning](#) on Wed, 24 Apr 2002 03:08:26 GMT

[View Forum Message](#) <> [Reply to Message](#)

Rick Towler (rtowler@u.washington.edu) writes:

- > To put it simply, when IDL renders a transparent atom it calculates the
- > color of the atom's surface as a function of it's own color, your alpha
- > value, and the color of objects behind it.

And other great stuff...

Thanks, Rick. I always hate these "which is in front of which" questions because about all I can do is wave my hands a bit and "mumble, mumble, right order, mumble" my way though it. It takes a heck of a lot of time to work your way though the details of this. I'm grateful for that wonderful pimento article (which I sometimes read when I want inspiration before going to bed) and for this great advice.

You really should work on an object book. :-)

Cheers,

David

--

David W. Fanning, Ph.D.

Fanning Software Consulting

Phone: 970-221-0438, E-mail: david@dfanning.com

Coyote's Guide to IDL Programming: <http://www.dfanning.com/>

Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: semi-transparent IDLgrPolygon? alpha blending no good...

Posted by [Sean Dettrick](#) on Wed, 24 Apr 2002 21:06:12 GMT

[View Forum Message](#) <> [Reply to Message](#)

Thanks David, Thanks Rick, your comments were both enlightening and practically very helpful.

I have another Q, but I'll post it seperately.

Cheers,

Sean Dettrick

Sean Dettrick wrote:

```
>
> (apologies for the attachment)
>
> Hi,
>
> Does anyone have a way to make semi-transparent polygon
> objects with IDLgrPolygon?
>
> I would like to use IDL's fancy object graphics, but in my
> attempts so far, alpha blending using a monochrome image only
> makes a polygon object transparent to ITSELF, not to other
> objects. This is pretty useless.
>
> Here is my example code, where I have rearranged
> David Fanning's object_shade_surface.pro (thanks DF)
> to instead plot up four cylindrical IDLgrPolygon objects.
> You can run it with:
> IDL> .r cylinder3
>
> I would like the red cylinder to be semi-transparent, so
> that I can see the black cylinder through it.
>
> I am away of Struan Gray's direct graphics tutorial,
> and guess one could do something similar with defining
> a z-buffer in the view object, but it is way above my head.
>
> Any assistance/pointers greatly appreciated.
>
> Thanks v much,
>
> Sean Dettrick,
> Physics Dept,
> University of California, Irvine
>
> -----
>
> ;+
> ; NAME:
> ;   OBJECT_SHADE_SURF
> ;
> ; PURPOSE:
> ;
> ;   This program shows you the correct way to write an
> ;   elevation-shaded surface in object graphics. This would
> ;   be the equivalent of these direct graphics commands:
> ;
```

```

> ;      Surface, data, Shades=BytScl(data)
> ;      Shade_Surf, data, Shades=BytScl(data)
> ;
> ; AUTHOR:
> ;
> ;      FANNING SOFTWARE CONSULTING
> ;      David Fanning, Ph.D.
> ;      2642 Bradbury Court
> ;      Fort Collins, CO 80521 USA
> ;      Phone: 970-221-0438
> ;      E-mail: davidf@dfanning.com
> ;      Coyote's Guide to IDL Programming: http://www.dfanning.com
> ;
> ; CATEGORY:
> ;
> ;      Object Graphics
> ;
> ; CALLING SEQUENCE:
> ;      OBJECT_SHADE_SURF, data, x, y
> ;
> ; INPUTS:
> ;      data: The 2D surface data.
> ;      x:   A vector of X values, corresponding to the X values of data.
> ;      y:   A vector of Y values, corresponding to the Y values of data.
> ;
> ; KEYWORD PARAMETERS:
> ;      STYLE: Set equal to 1 for a wire-frame surface. Set equal to 2 for
> ;             a solid surface (the default).
> ;
> ; COMMON BLOCKS:
> ;      None.
> ;
> ; EXAMPLE:
> ;      OBJECT_SHADE_SURF
> ;
> ; MODIFICATION HISTORY:
> ;      Written by: David Fanning, November 1998.
> ;
> ;
> FUNCTION Normalize, range, Position=position
>
> ; This is a utility routine to calculate the scaling vector
> ; required to position a vector of specified range at a
> ; specific position given in normalized coordinates. The
> ; scaling vector is given as a two-element array like this:
> ;
> ;      scalingVector = [translationFactor, scalingFactor]
> ;

```

```

> ; The scaling vector should be used with the [XYZ]COORD_CONV
> ; keywords of a graphics object or model. For example, if you
> ; wanted to scale an X axis into the data range of -0.5 to 0.5,
> ; you might type something like this:
> ;
> ; xAxis->GetProperty, Range=xRange
> ; xScale = Normalize(xRange, Position=[-0.5, 0.5])
> ; xAxis, XCoord_Conv=xScale
>
> On_Error, 1
> IF N_Params() EQ 0 THEN Message, 'Please pass range vector as argument.'
>
> IF (N_Elements(position) EQ 0) THEN position = [0.0, 1.0] ELSE $
>   position=Float(position)
> range = Float(range)
>
> scale = [((position[0]*range[1])-(position[1]*range[0])) / $
>   (range[1]-range[0]), (position[1]-position[0])/(range[1]-range[0])]
>
> RETURN, scale
> END
> ;-----
>
> Pro Object_Shade_Surf_Cleanup, tlb
>
> ; Come here when program dies. Free all created objects.
>
> Widget_Control, tlb, Get_UValue=info
> IF N_Elements(info) NE 0 THEN Obj_Destroy, info.thisContainer
> END
> ;-----
>
> PRO Object_Shade_Surf_Draw_Events, event
>
> ; Draw widget events handled here: expose events and trackball
> ; events. The trackball uses RSI-supplied TRACKBALL_DEFINE.PRO
> ; from the IDL50/examples/object directory.
>
> Widget_Control, event.top, Get_UValue=info, /No_Copy
>
> drawTypes = ['PRESS', 'RELEASE', 'MOTION', 'SCROLL', 'EXPOSE']
> thisEvent = drawTypes(event.type)
> dragQuality = 0
> CASE thisEvent OF
>
> 'EXPOSE': ; Nothing required except to draw the view.
> 'PRESS': BEGIN
>   Widget_Control, event.id, Draw_Motion_Events=1 ; Motion events ON.

```

```

> info.thisWindow->SetProperty, Quality=dragQuality ; Drag Quality to Low.
> needUpdate = info.thisTrackball->Update(event, Transform=thisTransform)
> IF needUpdate THEN BEGIN
>     info.thisModel->GetProperty, Transform=modelTransform
>     info.thisModel->SetProperty, Transform=modelTransform # thisTransform
> ENDIF
> END
> 'RELEASE': BEGIN
>     Widget_Control, event.id, Draw_Motion_Events=0 ; Motion events OFF.
>     info.thisWindow->SetProperty, Quality=2 ; Drag Quality to High.
>     needUpdate = info.thisTrackball->Update(event, Transform=thisTransform)
>     IF needUpdate THEN BEGIN
>         info.thisModel->GetProperty, Transform=modelTransform
>         info.thisModel->SetProperty, Transform=modelTransform # thisTransform
>     ENDIF
>     END
> 'MOTION': BEGIN ; Trackball events
>     needUpdate = info.thisTrackball->Update(event, Transform=thisTransform)
>     IF needUpdate THEN BEGIN
>         info.thisModel->GetProperty, Transform=modelTransform
>         info.thisModel->SetProperty, Transform=modelTransform # thisTransform
>     ENDIF
>     END
> ELSE:
>
> ENDCASE
>
> ; Draw the view.
>
> info.thisWindow->Draw, info.thisView
>
> ;Put the info structure back.
>
> Widget_Control, event.top, Set_UValue=info, /No_Copy
> END
> ;-----
>
> PRO Object_Shade_Surf_Resize, event
>
> ; The only events generated by this simple program are resize
> ; events, which are handled here.
>
> ; Get the info structure.
>
> Widget_Control, event.top, Get_UValue=info, /No_Copy
>
> ; Resize the draw widget.
>

```



```

> info.thisWindow->SetProperty, Dimension=[event.x, event.y]
>
>   ; Redisplay the graphic.
>
> info.thisWindow->Draw, info.thisView
>
>   ; Update the trackball objects location in the center of the
>   ; window.
>
> info.thisTrackball->Reset, [event.x/2, event.y/2], $
>   (event.y/2) < (event.x/2)
>
>   ;Put the info structure back.
>
> Widget_Control, event.top, Set_UValue=info, /No_Copy
> END
> ;-----
>
> PRO Object_Shade_Surf_Style, event
>
>   ; Change the style of the surface
>
> Widget_Control, event.top, Get_UValue=info, /No_Copy
> Widget_Control, event.id, Get_Value=thisButton
>
> CASE thisButton OF
>
>   'Wire Frame': info.thisSurface->SetProperty, Style=1
>   'Solid': info.thisSurface->SetProperty, Style=2
>
> ENDCASE
>
>   ; Draw the view.
>
> info.thisWindow->Draw, info.thisView
>
>   ;Put the info structure back.
>
> Widget_Control, event.top, Set_UValue=info, /No_Copy
> END
> ;-----
>
> PRO Object_Shade_Surf_Exit, event
>
>   ; Exit the program. This will cause the CLEANUP
>   ; routine to be called automatically.
>
> Widget_Control, event.top, /Destroy

```

```

> END
> ;-----
>
> PRO Object_Shade_Surf, thisSurface=thisSurface,surface2=surface2,$
>     surface3=surface3,surface4=surface4,light=light
>
>     ; Check for parameters.
>
> ;IF N_Elements(style) EQ 0 THEN style = 2
>
>     ; Create a view. Use RGB color. Charcoal background.
>
> thisView = OBJ_NEW('IDLgrView', Color=[80,80,80], $
>     Viewplane_Rect=[-1.0,-1.0,2.0,2.0])
>
>     ; Create a model for the surface and axes and add it to the view.
>     ; This model will rotate under the direction of the trackball object.
>
> thisModel = OBJ_NEW('IDLgrModel')
> thisView->Add, thisModel
>
>     ; Create an surface object shaded by elevation. Use Color Table 5.
>     ; Turn Gouraud shading on, but DON'T use a light source, as this will
>     ; modulate the shading parameters. Notice that adding a Palette object
>     ; to a surface object is NOT documented.
>
> ;numVerts = s[0]*s[1]
> ;thisPalette=Obj_New('IDLgrPalette')
> ;thisPalette->LoadCT, 5
>
> ;thisSurface = OBJ_NEW('IDLgrSurface', data, x, y, Style=style, Shading=1, $
> ;     Vert_Colors=Reform(ByteScl(data), numVerts), Palette=thisPalette)
>     ; Create axes objects for the surface. Color them green.
>
> xAxis = Obj_New("IDLgrAxis", 0, Color=[0,255,0], Ticklen=0.1, $
>     Minor=4, /Exact)
>
> yAxis = Obj_New("IDLgrAxis", 1, Color=[0,255,0], Ticklen=0.1, $
>     Minor=4, /Exact)
>
> zAxis = Obj_New("IDLgrAxis", 2, Color=[0,255,0], Ticklen=0.1, $
>     Minor=4)
>
>     ; Add the original surface and axes objects to the model.
>
> thisModel->Add, thisSurface
> ;thisModel->Add, xAxis
> ;thisModel->Add, yAxis

```

```

> ;thisModel->Add, zAxis
>
> ;add other surfaces and the light source
>
> if keyword_set(light) then thisModel->Add, light
> if keyword_set(surface2) then thisModel->Add, surface2
> if keyword_set(surface3) then thisModel->Add, surface3
> if keyword_set(surface4) then thisModel->Add, surface4
>
> ; Create a trackball for surface rotations. Center it in
> ; the window.
>
> thisTrackball = OBJ_NEW('Trackball', [200, 200], 200)
>
> ; Get the data ranges for the surface.
>
> thisSurface->GetProperty,XRange=xrange,YRange=yrange,ZRange=zrange
>
> ; Set scaling parameters for the surface and axes so that everything
> ; is scaled into the range -0.5 to 0.5. We do this so that when the
> ; surface is rotated we don't have to worry about translations. In
> ; other words, the rotations occur about the point (0,0,0).
>
> xs = Normalize(xrange, Position=[-0.5,0.5])
> ys = Normalize(yrange, Position=[-0.5,0.5])
> zs = Normalize(zrange, Position=[-0.5,0.5])
>
> ; Set the range, location, and scaling factors for the axes.
> ; Note that not all values in the Location keyword are
> ; used. (I've put really large values into the positions
> ; that are not being used to demonstrate this.) For
> ; example, with the X axis only the Y and Z locations are used.
>
> xAxis->SetProperty, Range=xrange, Location=[9999.0, -0.5, -0.5], $
> XCoord_Conv=xs
> yAxis->SetProperty, Range=yrange, Location=[-0.5, 9999.0, -0.5], $
> YCoord_Conv=ys
> zAxis->SetProperty, Range=zrange, Location=[-0.5, 0.5, 9999.0], $
> ZCoord_Conv=zs
>
> ; Scale the surfaces.
>
> thisSurface->SetProperty,XCoord_Conv=xs, YCoord_Conv=ys, ZCoord_Conv=zs
>
> ;if keyword_set(light) then $
> ;light->SetProperty,XCoord_Conv=xs, YCoord_Conv=ys, ZCoord_Conv=zs
> if keyword_set(surface2) then $
> surface2->SetProperty,XCoord_Conv=xs, YCoord_Conv=ys, ZCoord_Conv=zs

```

```

> if keyword_set(surface3) then $
> surface3->SetProperty,XCoord_Conv=xs, YCoord_Conv=ys, ZCoord_Conv=zs
> if keyword_set(surface4) then $
> surface4->SetProperty,XCoord_Conv=xs, YCoord_Conv=ys, ZCoord_Conv=zs
>
> ; Rotate the surface model to the standard surface view.
>
> ;thisModel->Rotate,[1,0,0], -90 ; To get the Z-axis vertical.
> thisModel->Rotate,[0,0,1], -90 ; Rotate it slightly to the right.
> thisModel->Rotate,[0,1,0], 60 ; Rotate it slightly to the right.
> ;thisModel->Rotate,[1,0,0], 30 ; Rotate it down slightly.
>
> ; Create the widgets to view the surface. Set expose events
> ; on the draw widget so that it refreshes itself whenever necessary.
> ; Button events are on to enable trackball movement.
>
> tlb = Widget_Base(Title='Elevation-Shaded Surface: Objects', Column=1, $
> MBar=menubase, TLB_Size_Events=1)
> drawID = Widget_Draw(tlb, XSize=400, YSize=400, Graphics_Level=2, Retain=0, $
> Expose_Events=1, Event_Pro='Object_Shade_Surf_Draw_Events', Button_Events=1)
>
> ; Create FILE menu.
>
> filer = Widget_Button(menubase, Value='File', /Menu)
> quitter = Widget_Button(filer, Value='Exit', $
> Event_Pro='Object_Shade_Surf_Exit')
>
> ; Create STYLE menu.
>
> styleID = Widget_Button(menubase, Value='Style', /Menu, $
> Event_Pro='Object_Shade_Surf_Style')
> button = Widget_Button(styleID, Value='Wire Frame')
> button = Widget_Button(styleID, Value='Solid')
>
> Widget_Control, tlb, /Realize
>
> ; Get the window destination object. The view will
> ; be drawn when the window is exposed.
>
> Widget_Control, drawID, Get_Value=thisWindow
>
> ; Create a container object to hold all the other
> ; objects. This will make it easy to free all the
> ; objects when we are finished with the program.
>
> thisContainer = Obj_New('IDL_Container')
>
> ; Add created objects to the container.

```

```

>
> thisContainer->Add, thisView
> thisContainer->Add, thisTrackball
> thisContainer->Add, xAxis
> thisContainer->Add, yAxis
> thisContainer->Add, zAxis
> thisContainer->Add, thisSurface
> if keyword_set(surface2) then thisContainer->Add, surface2
> if keyword_set(surface3) then thisContainer->Add, surface3
> if keyword_set(surface4) then thisContainer->Add, surface4
> if keyword_set(light) then thisContainer->Add, light
>
> thisContainer->Add, thisModel
> ;thisContainer->Add, thisPalette
>
> ; Get the current transformation matrix, so it can be restored.
>
> thisModel->GetProperty, Transform=origTransform
>
> ; Create an INFO structure to hold needed program information.
>
> info = { origTransform:origTransform, $ ; The transformation matrix.
>         thisContainer:thisContainer, $ ; The object container.
>         thisWindow:thisWindow, $ ; The window object.
>         thisSurface:thisSurface, $ ; The surface object.
>         thisTrackball:thisTrackball, $ ; The trackball object.
>         thisModel:thisModel, $ ; The model object.
>         xAxis:xAxis, $ ; The X Axis object.
>         yAxis:yAxis, $ ; The Y Axis object.
>         zAxis:zAxis, $ ; The Z Axis object.
>         thisView:thisView } ; The view object.
>
> ; Store the info structure in the UValue of the TLB.
>
> Widget_Control, tlb, Set_UValue=info, /No_Copy
>
> ; Call XManager. Set a cleanup routine so the objects
> ; can be freed upon exit from this program.
>
> XManager, 'Object_Shade_Surf', tlb, $
> Cleanup='Object_Shade_Surf_Cleanup', $
> Group_Leader=groupLeader, /No_Block, $
> Event_Handler='Object_Shade_Surf_Resize'
>
> END
> ;-----
>
> ; NB need to decide - is L=nz or L=nz-1? (cf mirrorfield.pro)

```

```

> ;pro cylinder2
>
> ;common grid ,nz,nr
> ;common mirror, b,B1
> nz=30
> nr=10
>
> ;----- -
> ; inner and outer cylinders built in same way:
>
> ; build x,y,z arrays
> ntheta=20
>
> z=findgen(nz) # replicate(1,ntheta)
> th=replicate(1,nz) # findgen(ntheta)
>
> TEXTURE_COORD = [[[z/(nz-1)],[[th/(ntheta-1)]]]]
> texture_coord = reform(texture_coord,2,nz*ntheta,/overwrite)
> th = th/(ntheta-1)*2*PI
>
> x=nr*cos(th)
> y=nr*sin(th)
>
> ; first set of vertices
> vert = [0,1,nz+1,nz]
> polygon=intarr(5*nz*ntheta)
> iv=0
> ; build list of vertices
> for k=0,ntheta-2 do begin
>   for j=0,nz-2 do begin
>     polygon(iv:iv+4) = [4,vert]
>     iv = iv+5
>     vert=vert+1
>   endfor
>   vert=vert+1
> endfor
>
> ;----- -
> ; make plain image to drape over main cylinder so can use alpha channel;
>
> alpha=50
> ; red = replicate(200,nz) # replicate(1,ntheta)
> ; green = replicate(20,nz) # replicate(1,ntheta)
> ; blue = replicate(20,nz) # replicate(1,ntheta)
> ; alpha = replicate(alpha,nz) # replicate(1,ntheta)
> red = replicate(20,2) # replicate(1,2)
> green = replicate(200,2) # replicate(1,2)
> blue = replicate(20,2) # replicate(1,2)

```

```

> alpha2 = replicate(alpha,2) # replicate(1,2)
> ; planar interleaving for image (interleave=2)
> drape = [[[red]],[[green]],[[blue]],[[alpha2]]]
> drape2 = [[[red]],[[blue]],[[green]],[[alpha2]]]
> help,red,drape
>
> ;texture_coord =
> ;-----
> ; mirror coils:
> ; inner radius rin, outer radius rout
> rin=nr*1.45
> rout=nr*1.5
> width=1
> ; build x,y,z arrays
> ntheta=20
>
> z2 = replicate(1,ntheta) # [0,width,width,0,0]
> th2 = findgen(ntheta)/(ntheta-1)*2*PI # replicate(1,5)
> r2 = replicate(1,ntheta) # [rout,rout,rin,rin,rout]
>
> x2=r2*cos(th2)
> y2=r2*sin(th2)
>
> ; first set of vertices
> vert = [0,1,ntheta+1,ntheta]
> polygon2=intarr(5*5*ntheta)
> iv=0
> ; build list of vertices
> for k=0,3 do begin
>   for j=0,ntheta-2 do begin
>     polygon2(iv:iv+4) = [4,vert]
>     iv = iv+5
>     vert=vert+1
>   endfor
>   vert=vert+1
> endfor
>
> ;-----
> ; now make some object arguments for the main drawing routine:
>
> ; Create a plain, semitransparent image to drape over the main
> ; cylinder:
>
> image = OBJ_NEW('IDLgrImage', drape, INTERLEAVE=2, blend_function=[3,1-alpha])
>
> ; Create a light source (an argument for object_shade_surf)
>
> light = Obj_New("IDLgrLight",color=lightcolor,type=1,$

```

```

>         direction=[1,1,0], location=[-2*nr,-2*nr,0] )
>
> ; Create a cylinder, polygon object:
>
> thisSurface = OBJ_NEW('IDLgrPolygon', x, y, z, Style=2, Shading=1, $
>         polygon=polygon, color=[200,20,20] )
> ;
>         texture_coord= texture_coord,$
> ;
>         texture_map=image )
>
> ; an inner cylinder, with the same polygon ordering:
>
> surface2 = OBJ_NEW('IDLgrPolygon', x*0.2, y*0.2, z, Style=2, Shading=1, $
>         polygon=polygon )
>
> ; a mirror coil
>
> surface3 = OBJ_NEW('IDLgrPolygon', x2, y2, z2, Style=2, Shading=1, $
>         polygon=polygon2, color=[20,200,20] )
>
> ; 2nd mirror coil
>
> surface4 = OBJ_NEW('IDLgrPolygon', x2, y2, z2+nz-width, Style=2, Shading=1, $
>         polygon=polygon2, color=[20,200,20] )
>
> Object_Shade_Surf, thisSurface=thisSurface,light=light,surface2=surface2,$
>     surface3=surface3,surface4=surface4
>
> end

```
