Subject: Vector output of idlgrpolygon models
Posted by D D on Tue, 08 Nov 2011 14:22:53 GMT
View Forum Message <> Reply to Message

Hi,

For a large document I'm currently writing I have the need to make several 3D diagrams. As my drawing skills aren't up to much I thought why not use IDL to generate the models programmatically, draw them in 3d and then save them to a postscript file or similar.

[For the question without context skip to >>>>>> below!]

The first part was fine and I generated some nice object graphics based models. Then came the problem, how do I save these in a vector format?

I played around with the idlgrclipboard and idlgrprinter objects. Using the printer object idlgrprinter I can produce vector files but only in black and white (i'm on unix based machines so IDL uses the xprinter system) and i'm not sure how to add another printer to do this in colour. I found that I could make colour vector files easily with idlgrclipboard however this is suboptimal. The vector files produced draw ALL polygons whether they are visible from the current view or not. This results in rather large postscript files meaning I can't use them as my final document would be too large.

Finally at work I've got access to idl v8.0, this brings the idlgrpdf output object so I thought i'd give that a go. This looked like my solution as the vector pdf's produced are small in size and full colour. Unfortunately there appear to be several polygons missing in the output, looking carefully it appears that for a few points polygons in the background are being drawn on top of the foreground polys.

It looks like postscript output via idlgrclipboard is my best bet if I can turn off polygons that aren't seen. I guess this reduces to a problem of vector plane intersection (albeit in a rather large loop) where I toggle the HIDE property based on number of intersections > 0
>>>> >>>>>>>>>> So here's my question:
Is there a simple way to get idl to remove/hide all polygons which can't be seen in the current view? I realise with modern hardware this is often not time efficient (i.e. takes longer to check if a poly can be seen then it does to just draw it anyway) so may not be built in but in my case (writing to a file) the one off cost of checking for hidden polygons is worth it for significant file size reduction.

Some alternative/intermediate questions:

i) Is there a routine anywhere to check if a polyline/vector intersects with a polygon object?
ii) Am I missing an obvious solution (which doesn't just save the figure in a raster format).
iii) A better question may be how to fix the pdf output/is this a known issue?
iv) In order to draw a polyline to intersect with a polygon I need two points, one is the point of interest. I guess the second is the eye position, is this correct? Where is the eye position? What about the different projection types?

If it helps with context the particular figure I'm working with involves ten nested toroidal surfaces with varying extent in toroidal angle. For this region a significant number of polygons can be hidden from view at a time.

Apologies for the rambling nature of this post, if any clarification is needed then please let me know.

Many thanks for any help!

Subject: Re: Vector output of idlgrpolygon models
Posted by D D on Wed, 09 Nov 2011 13:30:23 GMT

View Forum Message <> Reply to Message

On Nov 8, 7:58 pm, D D <d19997919j...@gmail.com> wrote:
> On Nov 8, 5:13 pm, Karl <Karl.W.Schu...@gmail.com> wrote:
>
>> You might try looking at the REJECT property in IDLgrPolygon. If your surfaces are defined correctly so that the normals are correct, REJECT can be set to prevent drawing the polygons that face away from the viewer. This might reduce the number of unwanted polygons you are dealing with.
>
>> Also might look at the VECT_SORTING keyword in IDLgrClipBoard::Draw(). Graphics hardware uses Z-buffers to take care of the hidden surface removal problem. The clipboard doesn't have such hardware and does coarse-grained sorting of the objects based on their depth instead. I can't remember if it uses the average Z of the entire IDLgrPolygon object, or the average Z of each triangle making up the polygon. If the latter is true, then that level of resolution may be good enough to sort out your surfaces. There will likely be problems where the toroids intersect, so look carefully there. The clipboard object won't split up intersecting triangles and draw just the visible pieces.
>
>> A more robust solution would be to use a BSP tree to sort them out and take care of splitting intersecting faces, but that is a lot of work.
>
> I activate hardware rendering by default on my draw widgets, IDL then
> sets them to software if the machine doesn't have suitable hardware.

> Indeed I would have thought that the hardware to file comparison would
> be the most likely place for a difference to occur.
>
> As most of my surfaces aren't closed setting the REJECT keyword to
> anything other than 0 means I lose half of my surface. Additionally
> many of the "hidden" polygons will have normals pointing towards (or
> away from) the camera. It's a shame because at first I thought REJECT
> was what I was after. It is an option if I enforce a fixed viewpoint
> but this is not great.
>
> VECT_SORTING again looks useful but only alters the order items are
> drawn, not if they are drawn.
>
> Currently I draw my surfaces as a single polygon object, I have tested
> splitting each surface into the individual polygons and redrawing.
> This makes the visualisation and manipulation a fair bit slower but
> still doesn't allow automated hidden object removal with vect_sorting
> or reject.
>
> One (probably idiotic) thought i've just had is if it's possible for a
> user to click in the window to select a polygon (i.e. IDL can take an
> x-y position and tell you what you've clicked on) then you could
> technically click every polygon you can see and mark it, then once
> you've covered the screen toggle all the unmarked polygons HIDE
> property. How you would go about this in practice i'm not sure,
> moreover I'd guess this is likely to take a considerable amount of
> time if one were to scan over the full window at the smallest
> resolution. I'm sure some optimisation could be made but this still
> sounds like it's probably not a good way to go (if it's even
> possible).
>
> Any other suggestions would be much appreciated but I think it looks
> like there's no simple way to achieve this. It will either involve
> writing a routine, putting up with large files or invoking an external
> tool (i'm trying hidden object removal with CorelDraw at the moment
> but it doesn't like the number of objects I think).
>
> Thanks,
> David
>
> (Going slightly off topic [well off IDL anyway] : Searching for a
> solution to this I came across the C library GL2PS which apparently
> does this hidden object removal and is a routine for creating
> Postscript files from opengl commands. As I have little knowledge of
> opengl and no knowledge of C i've not got anywhere with it but it
> would certainly could form a nice output object! There is optional
> support for it in Paraview [which can read vrml] so this is one
> possible route.)

Sorry to double post but I just thought I'd post my current "solution"
for anyone who searches these groups in the future.

My idea of using the select routine turns out to not be too idiotic
after all. The window object contains the method select which you
provide with a view or scene and an x-y position on the display, it
then returns you an ordered list of object reference to all the
objects (depending on if you pass a view or scene you get different
objects) which exist at that point, ordered by distance from viewer.
So it is possible to loop over the x-y values corresponding to the
window dimensions and build up a list of all objects which are at the
front at a certain point.

Without any effort a brute force approach is to simply sample every
pixel and keep a list of all polygons which appear at the front at
least once. The polygons not in the list can then be hidden/deleted
leaving just the visible polys! The downside of this is it can be
quite slow. First of all the simple way I have coded it involves
nested loops (with an if statement in the inner loop :S) which tend
not to be so good. Secondly I believe to work effectively each polygon
needs to represented by an individual polygon object which can be
painful to manipulate.

There is however scope for optimisation, other than improving the code
structure one can also think of taking an image of the view and
producing a mask where the pixel colour is equal to the background
colour and treating these positions as ignorable in the subsequent
select scan.

Some example code is given below if it's of use to anyone.

Thanks for all the help,
David

```
FUNCTION CULL,WIND,VIEW,STEP

;Wind : window object reference to scan
;view : View object referenc containing items of interest
;Step : Number of pixels to step over, defaults to 3

IF N_ELEMENTS(WIND) EQ 0 THEN BEGIN
   PRINT,"ERROR: Must pass wind and view."
   RETURN,-1
ENDIF
IF N_ELEMENTS(VIEW) EQ 0 THEN BEGIN
   PRINT,"ERROR: Must pass wind and view."
   RETURN,-1
ENDIF

IF N_ELEMENTS(STEP) EQ 0 THEN STEP=3

;Probe wind
wind->GetProperty,DIMENSIONS=DIM

;Make object array
OBJ=OBJARR(DIM)

;Calculate number of points in each direction
DIM2=FIX(DIM/STEP)

;Get total number of points
NPTS=(DIM2[0]*1L)*(DIM2[1]*1L)

;Print message about size
PRINT,"Window has dimensions "+STRING(DIM[0],'(I0)')+" by
"+STRING(DIM[1],'(I0)')
PRINT,"With a step size of "+STRING(STEP,'(I0)')
PRINT,"This corresponds to a grid of "+STRING(DIM2[0],'(I0)')+" by
"+STRING(DIM2[1],'(I0)')
PRINT,"Which is "+STRING(NPTS,'(I0)')+" points."

;Get current time
s1=SYSTIME(/SECONDS)


;Initialise counter
CT=0L
tav=0L

;Define format
form='($,a,a,a,a,a)'


FOR k=0,DIM[0]-1,STEP DO BEGIN
   FOR l=0,DIM[1]-1,STEP DO BEGIN
      ;Get time
      t1=SYSTIME(/SECONDS)

      ;Update counter
      CT=CT+1L

      ;Print counter message
      ;Form position
      pos=[k,l]
```

```
        ;Get selection
        sel=wind->Select(view,pos)

        ;Check selection is not empty
        IF OBJ_VALID(sel[0]) NE 0 THEN BEGIN
           obj[k,l]=sel[0]
        ENDIF

        ;Get time
        t2=SYSTIME(/SECONDS)

        ;Calculate average time
        tav=(tav*(CT-1L)+(t2-t1))/CT

        ;Make report message
        MESG="Time for loop : "+STRING(t2-t1,'(I0)')+" s "  ;Probably
reads as zero
        MESG=MESG+" Time remaining ~ "+STRING((NPTS-CT)*tav,'(F8.2)')
+" s"

        ;Update counter display
        PRINT,
form=form,MESG,STRING(CT,'(I0)'),"of",STRING(NPTS,'(I0)'),STRING( "15b)
    ENDFOR
ENDFOR

;Get final time
s2=SYSTIME(/SECONDS)

;Print total time
PRINT,"Total time taken : "+STRING(s1-s2,'(I0)')

;Return object array
RETURN,obj

END
```

Subject: Re: Vector output of idlgrpolygon models
Posted by Karl[1] on Wed, 09 Nov 2011 17:27:49 GMT

View Forum Message <> Reply to Message

I was going to point out that Select could be called for each pixel, but you figured that out :-).

But I also didn't point that out because I still don't think it is a perfect solution.

Did you actually send the resulting polygon list to vector output and inspect the results?   I think that rendering the resulting polygon list to the display would look pretty good, since you're getting help from the depth buffer. But that won't be the case on a vector device.

If two of your surfaces (toroids) intersect, there are going to be some faces that intersect with each other.

```
   EYE
    |
    V

  \    /
   \  /
    \/
^    \/
|    /\
Z   /  \
X - - >
```

Your algorithm would (correctly) select both surfaces and put them both in the list to send to the vector device. That device is going to render then both in the order you supply them and you'll see one surface or the other near the intersection. The part of the second surface to draw that is behind the first drawn surface will draw on top of the first surface, which is incorrect.

But perhaps your data does not do this and you may be fine.

The GL2PS approach is very interesting. It uses the same technique that the IDL Clipboard uses (OpenGL Feedback buffer) and sorts things in a similar way if you pick the simple sort option. I had also mentioned using BSP trees and I note that GL2PS offers a BSP tree sort option. This would address the problem I illustrated above with the intersecting surfaces. These two surfaces would each be split at the intersection line, resulting in four surfaces. The two "back" pieces would not be in the remaining list and would not draw, leaving the two "front" pieces to represent the correct rendering of the intersection. Might still be worth a look if you need this sort of precision.

Subject: Re: Vector output of idlgrpolygon models
Posted by D D on Wed, 09 Nov 2011 19:28:12 GMT

View Forum Message <> Reply to Message

On Nov 9, 5:27 pm, Karl <Karl.W.Schu...@gmail.com> wrote:
> I was going to point out that Select could be called for each pixel, but you figured that out :-).
>
> But I also didn't point that out because I still don't think it is a perfect solution.
>
> Did you actually send the resulting polygon list to vector output and inspect the results?   I think that rendering the resulting polygon list to the display would look pretty good, since you're getting help from the depth buffer. But that won't be the case on a vector device.
>

> If two of your surfaces (toroids) intersect, there are going to be some faces that intersect with each other.
```
>
>     EYE
>      |
>      V
>
>    \     /
>     \   /
>      \ /
> ^     \/
> |     /\
> Z    /  \
> X-->
>
```
> Your algorithm would (correctly) select both surfaces and put them both in the list to send to the vector device. That device is going to render then both in the order you supply them and you'll see one surface or the other near the intersection. The part of the second surface to draw that is behind the first drawn surface will draw on top of the first surface, which is incorrect.
>
> But perhaps your data does not do this and you may be fine.
>
> The GL2PS approach is very interesting. It uses the same technique that the IDL Clipboard uses (OpenGL Feedback buffer) and sorts things in a similar way if you pick the simple sort option. I had also mentioned using BSP trees and I note that GL2PS offers a BSP tree sort option. This would address the problem I illustrated above with the intersecting surfaces. These two surfaces would each be split at the intersection line, resulting in four surfaces. The two "back" pieces would not be in the remaining list and would not draw, leaving the two "front" pieces to represent the correct rendering of the intersection. Might still be worth a look if you need this sort of precision.

I've sent a couple of preliminary attempts to vector output and things
do seem to work quite well (the file size decreased by an order of
magnitude, and then some!) however I need to tweak my settings a bit I
think as currently things are a bit over excited and throw away a few
more polygons than needed. This should be solvable at the expense of
more cpu time.

One puzzle i'm having is to why the execution time of the function I
gave previously varies by an order of magnitude between machines. It's
possible my setup is at fault: one machine is a fedora box with
software rendering whilst the other is a virtual machine running
lubuntu on top of vista using hardware graphics. The virtual machine
is much slower for this (although much more responsive when
interacting with the widget plot).

GL2PS does indeed look interesting, i'd got 80% of the way to

compiling Paraview with GL2PS support enabled when I ran out of disk
space :S
I think a BSP tree type approach would be interesting, and certainly a
good project for me to try sometime but for now I'll stick with the
Select method as I can't afford the time (the long document I'm
writing is my thesis so trying to keep distractions to a minimum but
this IDL challenge keeps nagging at me!)

Thanks,
David

Subject: Re: Vector output of idlgrpolygon models
Posted by penteado on Mon, 21 Nov 2011 22:35:54 GMT

View Forum Message <> Reply to Message

I meant "problems with software rendering". Though there are plenty
with hardware rendering as well.

On Nov 21, 8:30 pm, Paulo Penteado <pp.pente...@gmail.com> wrote:
> I do remember an article mentioning some problems with hardware
> rendering:
>
> http://www.idlcoyote.com/ng_tips/render.php
>
> There was a problem with PS drawing order, logged years ago as CR ID
> 51003.
