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
```
