Subject: Re: Displaying three images simultaneously (using Object Graphics)
Posted by Karl Schultz on Tue, 08 Nov 2005 22:39:21 GMT
View Forum Message <> Reply to Message

On Tue, 08 Nov 2005 22:00:45 +0000, Dick Jackson wrote:

> "Dick Jackson" <dick@d-jackson.com> wrote in message
> news:PZ7cf.441815$1i.267001@pd7tw2no...
>> "Karl Schultz" <k____schultz@rsinc.com> wrote in message
>> news:pan.2005.11.08.19.05.56.672000@rsinc.com...
>>
>>> I should point out that starting with IDL 6.2, IDL renders images using
>>> texture-mapped polygons, doing some of the steps above for you. Further,
>>> there is a new TRANSFORM_MODE property that will treat the image as a
>>> polygon during transforms, instead of just transforming the opposite
>>> corners and making a new 2D box from the new corner locations.
>>
>> Well, how about that. Thanks for the tip, Karl. My new example would then
>> be:
>>
>> im1=Obj_New('IDLgrImage', BytScl(BIndGen(3, 10, 10)), Location=[-10,10], $
>> Transform_Mode=1)
>> im2=Obj_New('IDLgrImage', BytScl(RandomU(seed, 3, 10, 10)), $
>> Transform_Mode=1)
>> im3=Obj_New('IDLgrImage', 255B-BytScl(BIndGen(3, 10, 10)),
>> Location=[10,10], $
>> Transform_Mode=1)
>> myModel = Obj_New('IDLgrModel')
>> myModel->Add, [im1, im2, im3]
>> XObjView, myModel
>>
>> Another advantage of this is that we get away from the IDLgrPolygon
>> texture map's required use of image sizes that are a power of two (the
>> images were resampled to 16x16 in my previous example)
>
> Someone else at RSI has kindly cautioned me that the IDLgrImage won't give
> exactly the same behaviour as a texture-mapped IDLgrPolygon. The Image
> objects don't really "live" in the same 3-D space as Polygon objects.
> Rather, they are just drawn into the view in sequence along with the other
> objects, depending on the order they were added.
>
> This example (images all at Z=0, polygons at Z=-10, 0 and 10) shows the
> rather bewildering appearance that mixing these objects in one view can
> give:
>
> im1=Obj_New('IDLgrImage', BytScl(BIndGen(3, 10, 10)))
> im2=Obj_New('IDLgrImage', BytScl(RandomU(seed, 3, 10, 10)))
> im3=Obj_New('IDLgrImage', 255B-BytScl(BIndGen(3, 10, 10)))

```
>  poly1=Obj_New('IDLgrPolygon',
>  [[-10,10,-10],[0,10,-10],[0,20,-10],[-10,20,-10]], $
>              Color=[255,255,255], Texture_Map=im1, $
>              Texture_Coord=[[0,0],[1,0],[1,1],[0,1]])
>  poly2=Obj_New('IDLgrPolygon', [[0,0],[10,0],[10,10],[0,10]], $
>              Color=[255,255,255], Texture_Map=im2, $
>              Texture_Coord=[[0,0],[1,0],[1,1],[0,1]])
>  poly3=Obj_New('IDLgrPolygon', [[10,10,10],[20,10,10],[20,20,10],[10,20,10]],
>  $
>              Color=[255,255,255], Texture_Map=im3, $
>              Texture_Coord=[[0,0],[1,0],[1,1],[0,1]])
>  im4=Obj_New('IDLgrImage', BytScl(BIndGen(3, 10, 10)), Location=[-10,10], $
>         Transform_Mode=1)
>  im5=Obj_New('IDLgrImage', BytScl(RandomU(seed, 3, 10, 10)), $
>         Transform_Mode=1)
>  im6=Obj_New('IDLgrImage', 255B-BytScl(BIndGen(3, 10, 10)), Location=[10,10],
>  $
>         Transform_Mode=1)
>  myModel = Obj_New('IDLgrModel')
>  myModel->Add, [poly1, poly2, poly3, im4, im5, im6] ; Images show up in front
>  ;myModel->Add, [im4, im5, im6, poly1, poly2, poly3] ; Polygons show up in
>  front
>  XObjView, myModel
>
>  So, if you want images to look as if they are really in the same 3-D space
>  as other geometric objects, use texture-mapped IDLgrPolygons (or
>  IDLgrSurfaces, I suppose).
```

You could change the DEPTH_TEST_DISABLE property to make your images behave the same as other objects.

In the description for this property for IDLgrImage, we note:

"To preserve backward compatibility with previous versions of IDL the default value of this property is different from that of other graphic objects using this property."

For other objects, the default is OFF; for IDLgrImage it is ON, meaning that depth testing is disabled.

When we changed the image object to render as a texture-mapped polygon, we had to maintain the transform and depth-related behaviors with respect to IDL 6.1 for backwards compatibility.

That is, the "squishy" rotation and "painter's" depth buffer behavior had to be preserved, like them or not. So, the IDL 6.2 implementation turns off depth buffer processing by default when rendering an IDLgrImage, so that it "paints" the same way it used to, as a depth-agnostic pixel

primitive. (Image primitives are often considered as 'pixel primitives' in graphics systems and are treated differently than geometric primitives.)

But if you want to go ahead and treat the image as a geometric primitive, set DEPTH_TEST_DISABLE to 0 to turn on depth testing and set TRANSFORM_MODE to 1 so that it transforms like a polygon.

And the LOCATION property can take 3 elements, so you can position the image in Z and have it sort itself out against other geometry with depth testing.

So, you should still be able to get what you want without resorting to texture mapping onto a polygon or surface yourself.

Karl
