Subject: Re: pixmap drawables in Object Graphics?
Posted by karl_schultz on Tue, 18 Dec 2001 16:05:25 GMT
View Forum Message <> Reply to Message

"Martin Downing" <martin.downing@ntlworld.com> wrote in message
news:<82wT7.22741$4e3.3004657@news6-win.server.ntlworld.com>...
> Hi all,
>
> This is similar to previous queries on object graphics and pixmaps but I
> would appreciate running this by the experts.
> I am writing a program to fit projections of a 3d surface model to its
> silhouette in an image (e.g. a radiograph). This method allows an estimate
> of object position to be recovered from the knowledge of the object shape
> and the image. My 3d data is a triangulated mesh which can be best stored as
> a IDLgrPolygon object. This is attractive as you can then easily specify a
> graphics model to render the object at specific rotations, and projections
> of complicated polygon objects can then be drawn rapidly using OpenGL.
> However, as this is part of a fitting process, I then read the drawable back
> into an image buffer using say tvrd(), do some image processing to get a
> goodness of fit quantity and repeat until a sufficient fit is found. I do
> not need to see each projection in an exposed draw widget, but as far as I
> can gather, pixmaps are not implemented in object graphics. So as I see it,
> my only option using object graphics is to use normal draw widgets, which
> seems like overkill. Is this true and does anyone else have a better idea.?

It sounds like you want to do off-screen rendering and then perform image analysis on the result. Use IDLgrBuffer as the destination graphics object (instead of an IDLgrWindow). Fish the pixels back out with the IDLgrBuffer::Read method, which puts the pixels in an IDLgrImage object. You can then get the pixels out of the IDLgrImage object for analysis. All this can happen without drawing anything to the screen.

Something like:

```
oBuffer = obj_new('IDLgrBuffer')
oBuffer->Draw, oView
oImage = oBuffer->Read()
oImage->GetProperty, DATA=imageData
```

What happens under the covers is that IDL renders your scene into a completely device-independent off-screen frame buffer, using a software renderer.

Karl