Subject: Re: Can a CALL_EXTERNAL .dll create a window?
Posted by Karl Schultz on Fri, 29 Aug 2003 21:23:36 GMT
View Forum Message <> Reply to Message

"Matt Feinstein" <nospam@here.com> wrote in message
news:o3sukvkfiasbfisit3j594ckbjo9eedi4a@4ax.com...
> Hi all--
>
> I'm trying to write a CALL_EXTERNAL .dll that does off-screen
> hardware-assisted OpenGL rendering. My first try crashes IDL pretty
> much immediately, so I'm trying to eliminate possibilites for bugs.
> This tends to be difficult, since you can't run a .dll by itself... &
> it would be good if I could get some help in focussing my efforts on
> likely suspects.
>
> The first suspect I can think of is that I have to create a Win32
> window in the .dll. The reason I have to do this is that to get an
> off-screen hardware assisted rendering context one -has- to begin with
> an on-screen hardware assisted rendering context, which, in turn,
> means that you have to create a window. Is there a fatal difficulty in
> doing this in an IDL CALL_EXTERNAL .dll? Or, better, is there some
> combination of window properties that make it OK?
>
> Any help here would be appreciated.
>
> Matt Feinstein

One (rather different) approach is to create an object of some sort that is derived from an existing IDL graphics object and override its Draw method. You can override the Draw method in IDL .PRO code and have it do a CALL_EXTERNAL to your C code that calls OpenGL. At this point, there is already a window and GL context active that were created by IDL and are the window and GL context that your OpenGL calls will be directed to. Of course, this means you'll be using a window provided by IDL.

I know this works because I recently hacked up a class derived from IDLgrVolume that calls the Volume Graphics library (VGL - http://www.volumegraphics.com/products/vgl/) to render a volume, instead of using IDL's volume renderer. When I told the VGL to render, it just happily used its OpenGL calls in the IDL window and context. The results were pretty encouraging, but my time and trial VGL license expired. I've been thinking about tossing what I have into the user-conrtrib lib anyhow.

Most of the complexity in this experiment was in the VGL interface. It is pretty straightforward to just make a few GL calls, which is probably a better example anyway.

One thing to think about is that you don't know the current OpenGL state when you get to your code. If you change it too much, that could mess it up for IDL because IDL doesn't expect user code to be changing OpenGL state. So, that means saving and restoring any GL state that you change. (And this sort of thing is FAR from "supported", so I wouldn't build a lot of things on this approach.) Luckily, OpenGL has a pretty rich set of state saving tools.

As far as which class to override goes, you'd have to experiment a bit. If you pick IDLgrModel, you'll probably get a pretty clean state - that is, you won't be getting a state that is set up for any particular primitive and your risk of confusing IDL is less.

You mention "off-screen hardware assisted rendering". I'm not sure that you are going to have much luck here. Most hardware accelerators are wired pretty tightly to the video system's frame buffer. I've not heard of many hardware accelerators that can render into off-screen memory. Do you want to render into a region that is in the video memory, but not in a displayed window? I guess I'm not sure how you are going to convert your on-screen hardware rendering context into an off-screen one. I'm not sure that Microsoft OpenGL will use hardware rendering when drawing to a device-dependent bitmap. I know that it reverts to software when writing to a DIB.

You *should* be able to create your own GL context and drawing target and render into it from a DLL. If it were me, I'd make a small C dummy app that calls it in order to get it up and running. Then get IDL to call it.

You also may want to look at the Mesa graphics library, the software rendering OpenGL work-alike that IDL uses. There is a part of the library called OSMesa (Off-screen Mesa) that will do OpenGL rendering into a memory frame buffer. There will be some linkage issues to think about, since IDL already uses OpenGL and Mesa.

Karl