Subject: Re: memory consumption when drawing an idlgrscene object
Posted by Karl Schultz on Tue, 08 Jul 2003 14:33:00 GMT
View Forum Message <> Reply to Message

"Jan" <staffNOSPAM@fys.ku.dk> wrote in message
news:Pine.LNX.4.44.0307072320400.2931-100000@johansen.fys.ku .dk...
> Hi guys
>
> Thanks for your help. The problem seems to be fixed, so I will try to
> summarise a little bit:
>
> I am using IDL 5.6 for Linux. The way I checked the memory consumption was
> basically that I ran my program in IDL, and at the same time I run top to
> monitor the memory consumption. By running the program twice, once when
> drawing the scene object, and once without drawing the scene (but doing
> everything else), I observed the ~18 MB difference in memory consumption.
> Another option is to use IDLDE and go through the program step by step
> while observing the memory consumption in top. In any case, it was clear
> that the actual drawing of the scene would require a lot of memory.

Top is OK, but you should also investigate the memory() function built into
IDL.

> As suggested by some of you, I should make a test program. So I did, and
> then by stepping through that in IDLDE I got a good feeling for what is
> happening. Obviously, when drawing the scene object, a lot of memory might
> be required. When destroying the scene object, this memory is then
> released again, or at least most of it. Karl Schultz mentioned that IDL
> would cache some information, and I suppose this is what happens here. By
> redoing the scene object several times with different arrays, and drawing
> the scene each time, the memory consumption will more or less stabilize on
> some level.

This stabilizing is an important factor when conducting memory usage
investigations. Some of the tables inside IDL may grow slightly to reach
"high-water marks" in terms of variable and object utilization. One very
common practice used here at RSI is to run the code suspected of leaking
twice. The first time, ignore the memory measurements, and the second time
pay attention to them.

```
pro proc_under_test
   obj = OBJ_NEW("myobj")
   OBJ_DESTROY, obj
end

pro test
   proc_under_test
   mem1 = memory()
```

```
   proc_under_test
   mem2 = memory()
   print, mem1, mem2
end
```

In fact, to see what I am talking about a little easier, start a fresh IDL session and:

```
IDL> print, memory()
458966 378 132 460726
IDL> print, memory()
458990 380 133 458990
IDL> print, memory()
458990 381 134 458990
```

Note that there is a slight increase in memory between the first and second invocations and none between the second and third.

This is really important when analyzing memory problems, because a lot of people misinterpret the memory delta between the first and second invocations as a real problem.

> Also, one of my cleanup routines was not working properly, which meant
> that ~18 MB of memory was taken each time a scene was drawn. I did not
> look into that before, since I didn't consider it to be the problem.

Glad you found it.

> I still don't understand why an object all of a sudden takes up a lot of
> memory just by drawing it (and not when it is not drawn). Any ideas on
> that?

Sure. Because of the caches I mentioned. When a graphics object like IDLgrContour is instantiated, it really only stores the properties and data that you supply to it. In the case of IDLgrContour, this might be just the 2D array that you pass in as the first argument. Often, nothing else happens until you draw the object. At this point, IDL converts the data that you have supplied into a form that is optimized for rendering and stores this data into a cache that remains until the object (or window it was drawn on) is destroyed. This way, subsequent drawings of the scene containing the object go much faster. This improves the performance of animations and trackball manipulations.

The amount of benefit the caches provide depend on the object. IDLgrContour and IDLgrSurface benefit greatly because the caches require quite a bit of computation to compute - more than you would want to do on each redraw. In the case of IDLgrContour, the contouring algorithm is applied to generate the contour lines.

> Because of this, I suppose there is no way of seeing how much memory
> each object requires?

Not really. The caches can be of varying size, depending on the properties that are set and a bunch of other factors. Also, each object has its own caching policy - some may not even cache at all. The whole caching thing is private to the object implementation and should be invisible to the user, except for the memory consumption, as you have noticed. Unless there is an object implementation bug that causes a leak, a memory problem might lie elsewhere, as you have experienced.

> If you still want to see my test program, you are welcome, just post a
> message. But I think I figured out what happened to the memory, so there's
> nothing spooky here. Once again: thanks guys.

No need. But I think the memory() function will be more useful to you in future investigations.

Karl
