Subject: Re: IDL Objects Graphics cache and crash
Posted by Karl Schultz on Wed, 13 Mar 2002 16:14:43 GMT
View Forum Message <> Reply to Message

"Altyntsev Dmitriy" <alt@iszf.irk.ru> wrote in message
news:6b9fda50.0203130607.55173b3c@posting.google.com...

< snip (your're welcome) >

> One of the problems in this case is how to overlay different vector
> objects on image in most effective way. I mean when you change
> visibility (or position, color, one vertex coord...) of some layer the
> result of this change should be displayed as soon as possible. If it
> happens really fast it creates very good impression from work.
> Otherwise you just begin to avoid "clicking" because of latency. To
> illustrate the task, imagine the bundle of transparency films with
> pictures on them.
>
> In Direct Graphics, if some property is changed, the picture should be
> drawn from the beginning. I expected that Object Graphics is smarter
> in this case and caches "visual representation" as SOMETHING (e.g.
> images with mask or some image indexes to be drawn) and then renderer
> (hardware) sticks layers together very fast. But it seems OG draws the
> picture from the beginning too.

Yes, it does. The Draw method draws the entire tree without much
opportunity to do partial drawing. But read the section about instancing
below.

> Let's look at the small test that moves polyline over image.
>

```
> pro test
>    szx = 1000L
>    szy = 650L
>    img = bytscl(dist(szx,szy))
>    N = 100L
>    x = randomu(seed,N) * szx
>    y = randomu(seed,N) * szy
>
>    w = obj_new('IDLgrWindow', dim = [szx,szy], retain = 0, render = 1)
>    v = obj_new('IDLgrView', view = [0,0,szx,szy])
>    m = obj_new('IDLgrModel')
>    im = obj_new('IDLgrImage', img)
>    pl = obj_new('IDLgrPolyline', x, y, color = [255,0,0] )
>    v -> add, m
>    m -> add, im
>    m -> add, pl
>    t0 = systime(1)
>     for dx = 0, 100 do begin
>        pl -> SetProperty, xcoord = [dx,1]
>        w -> draw, v
>     endfor
>     print, 'Time = ', systime(1) - t0
>
>  end
>
```

> It takes 13.45 sec on my Pentium-III 700. It is very very slow. With
> render = 0 it is even slower. I am sure that modern video cards allow
> moving this polyline with the speed of bullet. Simple redrawing
> without moving takes the same time. It seems that image does not even
> moved as object in video memory. Probably because IDLgrImage is not
> actual 3D object. Or may be this entire task is not OpenGL purpose. Or
> I have some problems with my OpenGL driver? Although it works fine
> with outside OpenGL tests and games. time_test_gr2 seems to not work
> properly, but IDL demo works fine.

I go about twice as fast on my Pentium III 750 with software rendering. I may have a card/driver that is better at blits. I didn't even wait for the hardware renderer to finish, it was so slow.

Here's the deal:

OpenGL is not really great at drawing images in the sense of IDLgrImage. It has, for a long time, been optimized for fast polygonal rendering with texture mapping. Polygon fill rate was/is much more important than 2D image operations. All the slick OpenGL-based games you see out there are using texture-mapped polygons to draw the scenes.

Since gamers are the main users of 3D cards these days, the card vendors optimize the stuff important for games first. The code for handling 2D images is often very general and not optimized for obvious special cases. You may get lucky and stumble across a card/driver combination where the programmers spent some more time on the 2D paths. But in reality, the performance of the Quake benchmarks is the measuring stick.

From an IDL point of view, one might be tempted to employ Instancing to solve the problem. (See IDL docs for more information)

```
pro test2
   szx = 1000L
   szy = 650L
   img = bytscl(dist(szx,szy))
   N = 100L
   x = randomu(seed,N) * szx
   y = randomu(seed,N) * szy
```

```
  w = obj_new('IDLgrWindow', dim = [szx,szy], retain = 0, render = 1)
  v = obj_new('IDLgrView', view = [0,0,szx,szy])
  m = obj_new('IDLgrModel')
  im = obj_new('IDLgrImage', img)
  pl = obj_new('IDLgrPolyline', x, y, color = [255,0,0] )
  v -> add, m
  m -> add, im
  m -> add, pl

; hide changing part and draw static instance
  pl -> SetProperty, HIDE=1
  w -> draw, v, /CREATE_INSTANCE

; hide static part and start drawing dynamic part
  pl -> SetProperty, HIDE=0
  im -> SetProperty, HIDE=1
  v -> SetProperty, TRANSPARENT=1

  t0 = systime(1)
  for dx = 0, 100 do begin
    pl -> SetProperty, xcoord_conv = [dx,1]
    w -> draw, v, /DRAW_INSTANCE
  endfor
  print, 'Time = ', systime(1) - t0

end
```

This gives me about a 10% speedup on my machine, for the software renderer.
The hardware was still too slow. Instancing is MUCH more valuable when the
static part of the screen is really very complex. For example, if you have
a static "background" that is drawn with a large number of polygons and
other primitives and you wanted to move a few lines over it, as in this test
case, instancing can be a big win. Since instancing is implemented with 2D
image primitives anyway, using instancing to speed up a static scene
containing only an image will not make much of a difference, as I have
shown.

If your application has more in the static background portion than just an
image, please consider using instancing.

If you want to look at the texture mapping approach:

```
pro test3
  szx = 1000L
  szy = 650L
  img = bytscl(dist(szx,szy))
  N = 100L
  x = randomu(seed,N) * szx
```

```
   y = randomu(seed,N) * szy

   w = obj_new('IDLgrWindow', dim = [szx,szy], retain = 0, render = 0)
   v = obj_new('IDLgrView', view = [0,0,szx,szy])
   m = obj_new('IDLgrModel')
   im = obj_new('IDLgrImage', img)
   pl = obj_new('IDLgrPolyline', x, y, color = [255,0,0] )
   pg = obj_new('IDLgrPolygon',
[0,szx,szx,0],[0,0,szy,szy],[-0.1,-0.1,-0.1,-0.1], $
      color=[255,255,255], $
      TEXTURE_MAP=im, TEXTURE_COORD=[[0,0],[1,0],[1,1],[0,1]])
   v -> add, m
   m -> add, pg
   m -> add, pl
   t0 = systime(1)
   for dx = 0, 100 do begin
     pl -> SetProperty, xcoord = [dx,1]
     w -> draw, v
   endfor
   print, 'Time = ', systime(1) - t0

end
```

Note that I switched to the hardware renderer to take advantage of any texture acceleration. This code runs about 10% faster than test2 on my machine, so it would seem that this might be the best approach, especially if you are using a good 3D card, and if this test is close to your actual situation.

Your actual improvement will depend on the texture-mapped polygon fill rate of your card and perhaps the size of your texture memory.

> OG as a concept and utilities library is very very suitable for this
> task programming and it would be pity not to use it because of speed.
>
> I have feeling that I missed something very trivial. How can I make
> this stuff faster?

Hope some of these ideas help.
I know that a couple of 10% improvements don't help as much as you'd like, but we have to live with the limitations of the underlying graphics system sometimes.

Maybe someone with a different and better graphcs card can try these three programs and report the results. It may turn out that using a different card might be the best approach.

Karl