Subject: IDL and OPENGL
Posted by enric on Mon, 06 Sep 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Hi!

We are using IDL to develop a user interface. In this interface,
a graphical windows is opened and closed. This window constains a
IDL_Draw object quite big.
Our machine has a OPENGL graphics card. When we use the 'use hardware
OPENGL' flag... we loose 4 MB of memory each time the window is
opened/closed. If we set 'use software OPENGL' no memory is lost.

Anyone knows the reason/solution ??

Thanks,

Enric

Sent via Deja.com http://www.deja.com/
Share what you know. Learn what you don't.

Subject: Re: IDL and OpenGL
Posted by ushomirs on Wed, 29 Sep 1999 07:00:00 GMT
View Forum Message <> Reply to Message

> So are you saying when I am doing things like oWindow->Draw, oView and
the
> scene is full of 3D objects, Light objects, texture mapped images
etc. IDL
> should perform pretty well because this mostly happens in hardware via
> OpenGL lib calls (this is what I am hoping ? Upgraded graphics
hardware
> should meet our needs then - things like geometry and rasterization
accel in
> hardware chipsets etc.) But when it comes to pure number crunching
in the
> app, it may run a bit slower ?

yes, i think that's a fair way to paraphrase it.

Sent via Deja.com http://www.deja.com/
Before you buy.

Subject: Re: IDL and OpenGL
Posted by Richard Tyc on Wed, 29 Sep 1999 07:00:00 GMT
View Forum Message <> Reply to Message

ushomirs@my-deja.com wrote in message <7stme3$g15$1@nnrp1.deja.com>...
> I think what SGI folks meant is that IDL routines are **not** compiled
> into **machine** code, the way C programs are. So even an IDL routine
> is "compiled", it is transformed into a bytecode for it's own virtual
> machine. So unlike compiled C code, where the processor executes your
> program directly, the processor executes the code for the "IDL virtual
> machine", which in turn reads your program (or the "compiled" bytecode)
> and interprets it step by step. So there is an extra level of
> indirection involved in running IDL code. that's what the SGI people
> were refering to.
>
> If IDL code were truly compiled in the same sense as C code, then
> this loop
> FOR i=0,9 do foo[i]=2.*foo[i]
>
> would take exactly as long as
>
> foo=2.*foo
>
> It doesn't, because the IDL "virtual machine" has to interpret every
> iteration of the FOR loop, but it can just map the vector operation onto
> a machine code for loop.
>
> Now, in case of OpenGL, this distinction is a wash. So long as most of
> the time is spent inside OpenGL, some slowness associated with
> interpreting rather than executing machine code directly doesn't matter.
>
> greg

So are you saying when I am doing things like oWindow->Draw, oView and the
scene is full of 3D objects, Light objects, texture mapped images etc. IDL
should perform pretty well because this mostly happens in hardware via
OpenGL lib calls (this is what I am hoping ? Upgraded graphics hardware
should meet our needs then - things like geometry and rasterization accel in
hardware chipsets etc.) But when it comes to pure number crunching in the
app, it may run a bit slower ?

Thanks for the info. I am trying to understand the limitations of IDL.
Right now, we are running on fairly slow hardware (SGI O2 R5k) and I have
unacceptable 3D performance when it comes to realtime updates of Views with
complex volume objects being rendered.   It's far easier to sink $10-15K
down on faster hardware (dual CPU to benefit the IDLgrVolume object) then
start looking at a complete overhaul of our app using a different

development environment.

Rich

Subject: Re: IDL and OpenGL
Posted by ushomirs on Wed, 29 Sep 1999 07:00:00 GMT
View Forum Message <> Reply to Message

I think what SGI folks meant is that IDL routines are **not** compiled
into **machine** code, the way C programs are. So even an IDL routine
is "compiled", it is transformed into a bytecode for it's own virtual
machine. So unlike compiled C code, where the processor executes your
program directly, the processor executes the code for the "IDL virtual
machine", which in turn reads your program (or the "compiled" bytecode)
and interprets it step by step. So there is an extra level of
indirection involved in running IDL code. that's what the SGI people
were refering to.

If IDL code were truly compiled in the same sense as C code, then
this loop
FOR i=0,9 do foo[i]=2.*foo[i]

would take exactly as long as

foo=2.*foo

It doesn't, because the IDL "virtual machine" has to interpret every
iteration of the FOR loop, but it can just map the vector operation onto
a machine code for loop.

Now, in case of OpenGL, this distinction is a wash. So long as most of
the time is spent inside OpenGL, some slowness associated with
interpreting rather than executing machine code directly doesn't matter.

greg

THere's some discussion of this in IDL's external development guide.

In this respect IDL is much more like Java, where th

In article <37F22743.52AC5811@ssec.wisc.edu>,
  Liam Gumley <Liam.Gumley@ssec.wisc.edu> wrote:

> Richard Tyc wrote:
>> Our group was having a meeting with some SGI folks yesterday and
some
>> interesting points were brought up which I hope some IDL experts
could shed
>> some light on. We were discussing the performance of IDL vs. OpenGL
source
>> based app.. Their point was that IDL is an interpreted language and
as such
>> is not optimized (or tuned as they put it) for OpenGL and thus runs
>> significantly slower than a custom C app using OpenGL.
>
> IDL does indeed run in an interpreting mode (at the command line),
where
> each statement is interpreted and executed separately. However once
you
> 'compile' an IDL procedure, it is no longer 'interpreted' each time it
> is called.
>
> Perhaps someone in the know could give us a brief description of how
IDL
> procedures are transformed from source code to CPU instructions.
>
> Cheers,
> Liam.
>
> --
> Liam E. Gumley
> Space Science and Engineering Center, UW-Madison
> http://cimss.ssec.wisc.edu/~gumley
>

Sent via Deja.com http://www.deja.com/
Before you buy.

Subject: Re: IDL and OpenGL
Posted by Liam Gumley on Wed, 29 Sep 1999 07:00:00 GMT

View Forum Message <> Reply to Message

Richard Tyc wrote:
> Our group was having a meeting with some SGI folks yesterday and some
> interesting points were brought up which I hope some IDL experts could shed
> some light on. We were discussing the performance of IDL vs. OpenGL source
> based app.. Their point was that IDL is an interpreted language and as such
> is not optimized (or tuned as they put it) for OpenGL and thus runs
> significantly slower than a custom C app using OpenGL.

IDL does indeed run in an interpreting mode (at the command line), where
each statement is interpreted and executed separately. However once you
'compile' an IDL procedure, it is no longer 'interpreted' each time it
is called.

Perhaps someone in the know could give us a brief description of how IDL
procedures are transformed from source code to CPU instructions.

Cheers,
Liam.

--
Liam E. Gumley
Space Science and Engineering Center, UW-Madison
http://cimss.ssec.wisc.edu/~gumley

Subject: Re: IDL and OpenGL
Posted by Karl Schultz on Fri, 08 Oct 1999 07:00:00 GMT

View Forum Message <> Reply to Message

Richard Tyc wrote in message <7stqeb$jmk$1@canopus.cc.umanitoba.ca>...

> So are you saying when I am doing things like oWindow->Draw, oView and the
> scene is full of 3D objects, Light objects, texture mapped images etc.
IDL
> should perform pretty well because this mostly happens in hardware via
> OpenGL lib calls (this is what I am hoping ? Upgraded graphics hardware
> should meet our needs then - things like geometry and rasterization accel
in
> hardware chipsets etc.) But when it comes to pure number crunching in the
> app, it may run a bit slower ?

The IDL interpreter simply calls some compiled and optimized C code that
implements the Draw method. This code takes the graphic data associated
with the view and draws it with OpenGL as fast as it can manage. At this
point, the performance is bound mostly by the OpenGL implementation and any
hardware beneath it, since the view's graphic data is "ready for submission"
to the graphics system. The IDL interpeter gets control after the Draw()
method completes. So, yes, any improvement in graphics hardware should
improve graphics performance by about the same factor. This really only
applies if you don't pop back into IDL to do a lot of recalculation for each
graphic "frame".

If you have hardware-accelerated graphics, you may also see a net
improvement over the total time for running the app since the graphics

processors can offload the main CPU, giving it more time for other work.
This would especially be true of graphics systems that implement the
transformation pipeline in hardware.

> Thanks for the info. I am trying to understand the limitations of IDL.
> Right now, we are running on fairly slow hardware (SGI O2 R5k) and I have
> unacceptable 3D performance when it comes to realtime updates of Views with
> complex volume objects being rendered.   It's far easier to sink $10-15K
> down on faster hardware (dual CPU to benefit the IDLgrVolume object) then
> start looking at a complete overhaul of our app using a different
> development environment.

So true.

Subject: Re: IDL and OpenGL
Posted by David Fanning on Tue, 30 Oct 2007 12:50:22 GMT
View Forum Message <> Reply to Message

Gaurav writes:

> I would like to keep my discussion open ended and would welcome any
> advice related to the matter-be it OpenGL+IDL or makeing my program
> faster otherwise.

Well, I would scratch "Get one of those shiny new Macs"
off the list, at least for now. :-(

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: IDL and OpenGL
Posted by Robbie on Tue, 30 Oct 2007 22:17:54 GMT
View Forum Message <> Reply to Message

> exactly what tasks it does using OpenGL is not transparent
I've also had some difficulties optimising IDL's image tiling. I
wanted to place tiles off-screen so that they don't need to be fully

redrawn when they become visible. I kept finding that the off-screen tiles were not being stored in the GPU and I had to re-render them on demand.

I think that these kind of tricks are really beyond IDL at the moment. IDL provides a level of abstraction that is easy to program but difficult to optimise. If I wanted to render using openGL then I would write a C++ application using Coin3D. I would probably call that application as a DLM from IDL or insert it as an ActiveX component.

Robbie

Subject: Re: IDL and OpenGL
Posted by Gaurav on Wed, 31 Oct 2007 05:31:40 GMT
View Forum Message <> Reply to Message

Dear Dr. Fanning,
Thanks for the constant attention you bestow upon us paltry developers. But I did not quite get whatever it was you were trying to convey when you said:

"Well, I would scratch "Get one of those shiny new Macs"
off the list, at least for now. :-("

Would you please be a little elaborate because I am really stuck on this one and any good that might come out of this disussion will be for the greater good as far as I believe.

Cheers
Gaurav

Subject: Re: IDL and OpenGL
Posted by Gaurav on Wed, 31 Oct 2007 05:38:47 GMT
View Forum Message <> Reply to Message

That is exactly what I am afraid of doing right now. IDL takes care of other innuendo like event handling, mouse and keybord handling etc. with such ease that even to think of going to C or C++ to access the OpenGL libraries sends shudders down my spine.

But your suggestion about creating ActiveX component or creating DLM interests me quite a bit. I believe this will be the most simple way out. Could you be kind enough in pointing me towards some article that discusses such a concept in depth or a sample code that makes use of such a thing. I shall really appreciate the effort. And can I not

create the ActiveX component using say VB (cuz that is what I find
easier).

Cheers!
Gaurav

Subject: Re: IDL and OpenGL
Posted by David Fanning on Wed, 31 Oct 2007 12:31:41 GMT
View Forum Message <> Reply to Message

Gaurav writes:

> Thanks for the constant attention you bestow upon us paltry
> developers. But I did not quite get whatever it was you were trying to
> convey when you said:
>
> "Well, I would scratch "Get one of those shiny new Macs"
> off the list, at least for now. :-("
>
> Would you please be a little elaborate because I am really stuck on
> this one and any good that might come out of this disussion will be
> for the greater good as far as I believe.

I don't have anything to add to this discussion. (I pretty much
think what you want to do is hopeless in IDL.) My only point
is that when I think of making something faster in IDL
my list starts with a shiny new Mac. Recent discussion
suggests that IDL graphics and Leopard may be incompatible,
at least for now, so...

Cheers,

David

P.S. Hope you didn't think I was holding out on you,
or something. If I have something useful to say on a topic
I'm normally not shy about sharing it. :-)

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: IDL and OpenGL

Posted by Karl Schultz on Wed, 31 Oct 2007 16:29:49 GMT
View Forum Message <> Reply to Message

It would probably be wise to profile your application to determine
where the time is being spent. It would be unwise to go through
a lot of effort to replace the display subsystem when the problem
might be elsewhere.

You can use IDL's profiler or use SYSTIME to check suspected hotspots.

If the Window Draw method ends up being the one using all the time, then
perhaps you are sending too much geometry or have too complex of a scene.
You might consider adjusting the fineness of your meshes according
to the view.

If tiling is the bottleneck, consider using a tiling server running
in another process. This will at least let the main app run smoother
without stalling while waiting for tiles.

Object Graphics is a fairly thin layer over OpenGL and so there are
not many places where IDL slows things down for OpenGL. But you might
be able to do better by accessing some of OpenGL's features that IDL
doesn't use. Compiled vertex lists (VBO's) are an example, but I
do not know if IDL uses them or not.

Finally, and this is NOT recommended, you can make OpenGL calls using
the same IDLgrWindow and OpenGL context via C code. You start by making
a subclass of IDLgrModel and override the Draw method. Your Draw method
can then call C code that can make OpenGL drawing calls. The dicey part
of this is that the IDL internals can change from release to release
and something like this that worked in one release may not in another.
But it is a good way to prototype something to see if you are on the
right path.

Karl

Gaurav <selfishgaurav@gmail.com> wrote:
> Hi all,
> I have been using IDL to develop a Google Earth type application for
> the past one year and have been pretty successful in doing so. But the
> application is not as fast as I would like it to be. I believe I have
> removed all the programming bottlenecks that might slow my program
> down(Avoiding loops, using small tiles of images etc..ad infinitum). I
> was wondering if leaning more on OpenGL might be of some help.
>
> Now, I am aware that IDL already uses OpenGL for many of its tasks-but
> exactly what tasks it does using OpenGL is not transparent (at least
> to me). Can anyone suggest me ways to get on if I want to acess the
> OpenGL library of functions using IDL-the way they do it using C, C++,

> VB or VB.NET.
>
> I would like to keep my discussion open ended and would welcome any
> advice related to the matter-be it OpenGL+IDL or makeing my program
> faster otherwise.
>
> Cheers!
> Gaurav
>

--
Karl Schultz   kws@frii.com
There are 844,739 ways to enjoy a Waffle House hamburger.

Subject: Re: IDL and OpenGL
Posted by Steve Houston on Wed, 07 Nov 2007 14:28:42 GMT
View Forum Message <> Reply to Message

Karl Schultz wrote:
> It would probably be wise to profile your application to determine
> where the time is being spent. It would be unwise to go through
> a lot of effort to replace the display subsystem when the problem
> might be elsewhere.
>
> You can use IDL's profiler or use SYSTIME to check suspected hotspots.

I would like to second Karl's suggestion to use the profiling tools in
IDL to determine where the bottlenecks are.

>
> If the Window Draw method ends up being the one using all the time, then
> perhaps you are sending too much geometry or have too complex of a scene.
> You might consider adjusting the fineness of your meshes according
> to the view.
>
> If tiling is the bottleneck, consider using a tiling server running
> in another process. This will at least let the main app run smoother
> without stalling while waiting for tiles.
>
> Object Graphics is a fairly thin layer over OpenGL and so there are
> not many places where IDL slows things down for OpenGL. But you might
> be able to do better by accessing some of OpenGL's features that IDL
> doesn't use. Compiled vertex lists (VBO's) are an example, but I
> do not know if IDL uses them or not.

IDL 6.4 and later does use OpenGL vertex buffer objects (VBOs) to store
vertices and indices in video RAM, under most circumstances. This

greatly improves rendering performance as long as your vertices are
mainly static (you aren't providing new vertex data every time you
render the object).

>
> Finally, and this is NOT recommended, you can make OpenGL calls using
> the same IDLgrWindow and OpenGL context via C code. You start by making
> a subclass of IDLgrModel and override the Draw method. Your Draw method
> can then call C code that can make OpenGL drawing calls. The dicey part
> of this is that the IDL internals can change from release to release
> and something like this that worked in one release may not in another.
> But it is a good way to prototype something to see if you are on the
> right path.
>
> Karl
>
> Gaurav <selfishgaurav@gmail.com> wrote:
>> Hi all,
>> I have been using IDL to develop a Google Earth type application for
>> the past one year and have been pretty successful in doing so. But the
>> application is not as fast as I would like it to be. I believe I have
>> removed all the programming bottlenecks that might slow my program
>> down(Avoiding loops, using small tiles of images etc..ad infinitum). I
>> was wondering if leaning more on OpenGL might be of some help.
>>
>> Now, I am aware that IDL already uses OpenGL for many of its tasks-but
>> exactly what tasks it does using OpenGL is not transparent (at least
>> to me). Can anyone suggest me ways to get on if I want to acess the
>> OpenGL library of functions using IDL-the way they do it using C, C++,
>> VB or VB.NET.
>>
>> I would like to keep my discussion open ended and would welcome any
>> advice related to the matter-be it OpenGL+IDL or makeing my program
>> faster otherwise.
>>
>> Cheers!
>> Gaurav
>>
>

Subject: Re: IDL and OpenGL
Posted by Steve Houston on Wed, 07 Nov 2007 14:52:25 GMT
View Forum Message <> Reply to Message

Robbie wrote:
>> exactly what tasks it does using OpenGL is not transparent
> I've also had some difficulties optimising IDL's image tiling. I

> wanted to place tiles off-screen so that they don't need to be fully
> redrawn when they become visible. I kept finding that the off-screen
> tiles were not being stored in the GPU and I had to re-render them on
> demand.

Could you explain in more detail what you are doing?

When you call SetTileData for a tile that is offscreen the data is
loaded into a texture map and stored in video RAM. However, any tiles
that are offscreen will not be rendered, OpenGL clips everything to the
view frustum, so anything offscreen is essentially ignored.

When your tile comes onscreen, it will need to be rendered, but this
should be extremely fast as the texture data is already loaded into
video RAM, ready to go.

> I think that these kind of tricks are really beyond IDL at the moment.
> IDL provides a level of abstraction that is easy to program but
> difficult to optimise.

IDL is designed to hide the details of the underlying rendering API, to
make it easier to program and so a different renderer can be used if
necessary. It does this while adding as little overhead to the
underlying rendering API as possible.

The IDL object graphics API is designed to be as flexible as possible,
but it's not as low level as OpenGL, so there will be some things you
can do in an application calling OpenGL directly that you can't in IDL.
If this is the case I recommend you contact ITTVIS or post in this
newsgroup so we can evaluate the functionality you require.

> If I wanted to render using openGL then I would
> write a C++ application using Coin3D. I would probably call that
> application as a DLM from IDL or insert it as an ActiveX component.
>
> Robbie
>

Steve.