Subject: Re: IDLgrWindow, IDLgrVolume and alpha channel (more details)
Posted by s[1] on Mon, 27 Jan 2003 09:14:16 GMT
View Forum Message <> Reply to Message

Hi all,

first of all, thanks for all infos.
Here come some more details:
I am rendering two volumes, similar to the MRI + PET demo, but its two
volumes (and not two data sets in one volume) and the zbuffer of the first
is intersected with the second one.
I am rendering the two volumes to two different IDLgrBuffers.
The resulting images are then blended.

I need the alpha for two reasons:
a) To determine the contribution of each image.
b) To find the image background (o.k., that I can also do with the
zbuffer).

Another, not so important reason is that I am thinking about using the
alpha values from the image to adapt the transfer function.

I was already thinking about the "linear ramp and greyscale" solution, but
for some volumes I have to use a non-linear opacity function, so that
doesn't work.

For me, getting the 4-channel IDLgrImage Karl was talking about would be
the perfect solution.
Do I understand you correctly that there is no chance to get the alpha?
Is there a difference between using a IDLgrBuffer and IDLgrWindow for this
problem?
I think having the alpha channel during rendering and just throwing it
away without having the least chance to get it is just stupid - there
will always be someone who might make use of it...

Best regards & thanks for all help,

Sebastian

Subject: Re: IDLgrWindow, IDLgrVolume and alpha channel (more details)
Posted by Karl Schultz on Tue, 28 Jan 2003 14:58:47 GMT
View Forum Message <> Reply to Message

"Sebastian" <s@visita2.die.upm.es> wrote in message
news:Pine.LNX.4.44.0301270946510.1959-100000@visita2.die.upm .es...
> Hi all,
>

> first of all, thanks for all infos.
> Here come some more details:
> I am rendering two volumes, similar to the MRI + PET demo, but its two
> volumes (and not two data sets in one volume) and the zbuffer of the first
> is intersected with the second one.
> I am rendering the two volumes to two different IDLgrBuffers.
> The resulting images are then blended.
>
> I need the alpha for two reasons:
> a) To determine the contribution of each image.
> b) To find the image background (o.k., that I can also do with the
> zbuffer).
>
> Another, not so important reason is that I am thinking about using the
> alpha values from the image to adapt the transfer function.
>
> I was already thinking about the "linear ramp and greyscale" solution, but
> for some volumes I have to use a non-linear opacity function, so that
> doesn't work.

OK, this helps us understand the need for this.

> For me, getting the 4-channel IDLgrImage Karl was talking about would be
> the perfect solution.

This isn't that hard to do. I'll open a change request.

> Do I understand you correctly that there is no chance to get the alpha?

Yes.

> Is there a difference between using a IDLgrBuffer and IDLgrWindow for this
> problem?

No. Both IDLgrBuffer::Read and IDLgrWindow::Read return RGB arrays. I suppose that IDLgrBuffer could be fixed up to use destination alpha and return RGBA with Read, but I think that getting the data from IDLgrVolume addresses your need more directly. Remember that the destination alpha values you would pull out of the frame buffer may not correspond directly to the source alpha values of your primitive (volume). They might, if the blend equations were just right and there were no other primitives in the scene. Anyway, my point is that trying to derive the original alpha values by getting them from the device frame buffer is a roundabout way of doing it.

> I think having the alpha channel during rendering and just throwing it
> away without having the least chance to get it is just stupid - there
> will always be someone who might make use of it...

Yeah, and IDLgrVolume itself could make use of it if it knew that it could
redraw a volume with the cached image data. To explain this statement, I
need to explain a bit more about how object graphics caching works.

IDLgrSurface, for example, takes your 2D Z data and creates a polygonal mesh
when it is drawn for the first time. After the draw, the mesh is cached
because it can be reused on subsequent draws, even if some drawing
attributes change. If the current transform and/or surface color changes,
the cache can be reused because the transform and color are not part of the
cache and are bound to the surface at draw time.

IDLgrVolume would not be able to reuse a cached image as easily because the
image contents depend a great deal on many graphic properties, including
transforms. The IDLgrVolume object would have to also record all the state
that applied when creating the cached image and check the current state
against the recorded state to see if the cached image is ok to use to draw
the volume. Because of the expense and peril of performing this check and
the *relatively* lower likelihood of the state staying static enough to
reuse the image, the current implementation doesn't cache the image. But
maybe we can look into this.

Karl

Subject: Re: IDLgrWindow, IDLgrVolume and alpha channel (more details)
Posted by Rick Towler on Thu, 30 Jan 2003 18:21:19 GMT
View Forum Message <> Reply to Message

"Karl Schultz" wrote >

> "Sebastian" wrote

>> Is there a difference between using a IDLgrBuffer and IDLgrWindow for
this
>> problem?
>
> No. Both IDLgrBuffer::Read and IDLgrWindow::Read return RGB arrays. I
> suppose that IDLgrBuffer could be fixed up to use destination alpha and
> return RGBA with Read, but I think that getting the data from IDLgrVolume
> addresses your need more directly. Remember that the destination alpha
> values you would pull out of the frame buffer may not correspond directly
to
> the source alpha values of your primitive (volume). They might, if the
> blend equations were just right and there were no other primitives in the
> scene. Anyway, my point is that trying to derive the original alpha
values
> by getting them from the device frame buffer is a roundabout way of doing

> it.

If we're throwing around feature requests I want to get in!

I think that fixing IDLgrBuffer to use destination alpha is a good idea.
Recently I have been procedurally creating textures (rendering to a buffer
then texmapping) and although I haven't had a need for the alpha channel
yet, I can see the possibilities.

But I wouldn't want this to get in the way of that new renderer you are
working on. You know, the one that handles order-independent transparency?
:)

-Rick