Subject: Re: texture map irregularities OR pimento problems
Posted by david[2] on Fri, 22 Jun 2001 00:14:03 GMT
View Forum Message <> Reply to Message

Rick Towler writes:

> I have been experimenting using the alpha channel to represent confidence in
> a data set. I produce a polygon object representing the data and then
> texture map the polygon accordingly. But, I have run into an issue that I
> can't resolve.

I thought the exam for the IDL Expert Programmers Association
recruits was in September. When did it get moved up to June!?

Cheers,

David

P.S. Let's just say I'm glad I got grandfathered in, because
I don't think I could make it anymore. :-(

--
David Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: texture map irregularities OR pimento problems
Posted by Rick Towler on Fri, 22 Jun 2001 02:15:39 GMT
View Forum Message <> Reply to Message

Update: On another machine with the same graphics adapter but different
driver revision the first case renders exactly like the software renderer.
Which is good, I guess (but this is actually farther away from what I want).
I still don't understand why I can't see the inside of the green orb in the
first case.

-Rick

"Rick Towler" <rtowler@u.washington.edu> wrote in message
news:9gtvnl$q08$1@nntp6.u.washington.edu...
> I have been experimenting using the alpha channel to represent confidence
in
> a data set. I produce a polygon object representing the data and then
> texture map the polygon accordingly. But, I have run into an issue that I

> can't resolve.
>
> I have attached a simple example (requires at least IDL 5 something and 24
> bit display). I create 2 orb objects, one green (the olive) and one red
> (the pimento).
>
> At first I place the olive in front of the pimento, then I make the near
> surface of the olive transparent. Rendering in hardware for the most part
> works except between the last opaque vertices and the first transparent
> vertices I get a white (background color) ring where you can actually see
> thru the entire olive to the pimento behind it. If your imagination is
> failing I'll remind you that we are just supposed to be looking into the
> inside of the olive, we shouldn't actually see the pimento thru the olive.
> Rendering in software, things get even more wacky. I don't see the inside
> of my olive at all. Xray vision I guess. What is interesting to note is
> that this ring I get when rendering in hardware renders just like the all
of
> the transparent vertices in software.
>
>
> I then flip the pimento and the hole in my olive. The pimento now is in
> front of the olive and the "hole" is created on the back side of the green
> orb. Now when I rotate the olive around so I can see in my hole, things
> look pretty good. The astute observer will notice that the ring is still
> there (you will see a little red pimple if viewed from the right angle)
but
> otherwise things look good. In software mode, things seem to work.
>
>
> What is going on in the first case? Why does software rendering fail to
> render the inside of the olive? What am I missing?
>
> I would be happy to provide images upon request.
>
> -Rick Towler
>
>
```
> ;pimento example
>
> container = obj_new('IDL_Container')
>
> ;create a rainbow texture map with varying alpha
> palette = obj_new('IDLgrPalette')
> palette -> LoadCT, 13
> container->add, palette
>
> imagedata = bytarr(4, 256, 256, /nozero)
> for a = 0, 255 do begin
>     for nc=0,255 do begin
>         imagedata[0:2,a,nc] = palette -> GetRGB(nc)
>         imagedata[3,a,nc] = a
>     endfor
>  endfor
>  texmap = obj_new('IDLgrImage', imagedata, interleave=0, $
>         blend_function=[3,4], /interpolate)
>  container-> add, texmap
>
>
>  ;a model to stick things in
>  model = obj_new('idlgrmodel')
>  container -> add, model
>
>
>  ;some reference orbs
>  pimento = obj_new('Orb', pos=[0,0,-3.0], radius=1.0, color=[255,0,0],
>  density=2.0)
>  container-> add, pimento
>
>
>  ;something to texmap
>  olive = obj_new('Orb', pos=[0,0,0], radius=2.0, color=[255,255,255],
>  density=2.0)
>  container-> add, olive
>
>
>  ;add atoms from back to front
>  model -> add, [pimento, olive]
>
>  ;get the olive's vertex data
>  olive -> getproperty, data=vtex
>
>
>  ;calculate an array of latitudes for each vertex point
>  vtex_dims = size(vtex, /dimensions)
>  latitude = fltarr(vtex_dims[1])
>  nf = sqrt(total(vtex^2,1))
>  for n = 0, vtex_dims[1] - 1 do begin
>     vtex_norm = vtex[*,n] / nf[n]
>     latitude[n] = acos(vtex_norm[2] / sqrt(total(vtex_norm^2.))) * !RADEG
>  endfor
>
>
>  ;set the texture coordinates
>  ;create a "bowl" by making the positive z verticies with latitudes
>  ;less than 45 deg transparent.  The result is an orb with a hole in
>  ;the side.
>  texcoords = fltarr(2,vtex_dims[1], /nozero)
>  texcoords[1,*] = .65  ;pick a nice green color for our olive
>  texcoords[0,*] = 0.99
>  ;set the opacities to ~0.0 at lats lt 45.
>  i = where(latitude lt 45.)
>  texcoords[0,i] = 0.01
>
>
>  ;set some properties
>  ;note that you have to set reject=0 since by default orbs set it to 1
>  olive -> setproperty, texture_map=texmap, texture_coord=texcoords, $
>      /texture_interp, /zero_opacity_skip, reject=0
>
>
>  ;take a look at what we got
>  xobjview, model, /block
>
>
>  ;now flip this whole deal in the z direction
>
>  ;move our pimento
>  pimento -> setproperty, pos=[0,0,3.0]
>
>
>  ;cut the top off of the other side of the bowl where
>  ;latitude is gt 180. - 45.
>  texcoords[0,*] = 0.99
>  i = where(latitude gt 180. - 45.)
>  texcoords[0,i] = 0.01
>
>  olive -> setproperty, texture_coord=texcoords
>
>
>  xobjview, model, /block
>
>  ;cleanup
>  obj_destroy, container
>
>  end
>
>
```

Subject: Re: texture map irregularities OR pimento problems
Posted by david[2] on Fri, 22 Jun 2001 04:28:09 GMT
View Forum Message <> Reply to Message

Rick Towler writes:

> Update: On another machine with the same graphics adapter but different
> driver revision the first case renders exactly like the software renderer.
> Which is good, I guess (but this is actually farther away from what I want).
> I still don't understand why I can't see the inside of the green orb in the
> first case.

I'm not at all surprised by the different hardware
rendering. Every machine (practically) that I have
ever run an object graphics program on with hardware
rending has rendered the scene differently. Unless I
want trouble, I always choose software rendering for
the draw widget in object graphics programs. I spent
too much time when I released my first object programs
chasing phantom bugs. :-(

But I don't have a clue about the rest of it. Maybe
Randy Frank is listening in. :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: texture map irregularities OR pimento problems
Posted by Karl Schultz on Fri, 22 Jun 2001 16:52:18 GMT

View Forum Message <> Reply to Message

The order in which the polygons are drawn is extremely important when using
alpha blending.

In the first case:

1) The red orb is drawn first without alpha blending. So far, so good.

2) The green orb is then drawn with alpha blending. You don't know
beforehand
which polygons in the green orb draw first. What is happening is that the
top
(pos z) part of the orb is drawing first. This is the transparent part, and
after we draw

this part, we see the red orb and parts of the background, as we expect.

Now drawing these transparent
polygons also sets the Z buffer in this part of the screen. So, as we draw
the rest of
the green orb, the bottom or -Z part, the Z buffer test fails and these
polys do not
draw. The part of the screen where the transparent polys are does not
change, and
you end up seeing background and red orb where you instead
wanted to see the inside of the green orb.

In the second case:

- The transparent part of the green orb is drawn last, because you moved it
from
+z to -z, in terms of the orb coords. (Assuming that you've spun it around
180 in xobjview)
So, the side opposite the transparent "hole" draws first since it is in the
+z
part of the orb. This parts makes it onto the screen and then the
transparent polygons are drawn on top, letting you see the inside.

Hope this helps. The general point is that you really have to be very
careful in
layering primitives when drawing with alpha. The layering may need to take
into account the viewing angle and the geometry of the primitives.

For example, in this case, you may have to rotate the orb so that it always
draws
back-to-front and move the transparent part so that it is always "in front",
with
respect to the viewing direction.

Karl

"Rick Towler" <rtowler@u.washington.edu> wrote in message
news:9gtvnl$q08$1@nntp6.u.washington.edu...
> I have been experimenting using the alpha channel to represent confidence
in
> a data set. I produce a polygon object representing the data and then
> texture map the polygon accordingly. But, I have run into an issue that I
> can't resolve.
>
> I have attached a simple example (requires at least IDL 5 something and 24
> bit display). I create 2 orb objects, one green (the olive) and one red
> (the pimento).

>
> At first I place the olive in front of the pimento, then I make the near
> surface of the olive transparent. Rendering in hardware for the most part
> works except between the last opaque vertices and the first transparent
> vertices I get a white (background color) ring where you can actually see
> thru the entire olive to the pimento behind it. If your imagination is
> failing I'll remind you that we are just supposed to be looking into the
> inside of the olive, we shouldn't actually see the pimento thru the olive.
> Rendering in software, things get even more wacky. I don't see the inside
> of my olive at all. Xray vision I guess. What is interesting to note is
> that this ring I get when rendering in hardware renders just like the all
of
> the transparent vertices in software.
>
>
> I then flip the pimento and the hole in my olive. The pimento now is in
> front of the olive and the "hole" is created on the back side of the green
> orb. Now when I rotate the olive around so I can see in my hole, things
> look pretty good. The astute observer will notice that the ring is still
> there (you will see a little red pimple if viewed from the right angle)
but
> otherwise things look good. In software mode, things seem to work.
>
>
> What is going on in the first case? Why does software rendering fail to
> render the inside of the olive? What am I missing?
>
> I would be happy to provide images upon request.
>
> -Rick Towler

Subject: Re: texture map irregularities OR pimento problems
Posted by Rick Towler on Fri, 22 Jun 2001 20:35:00 GMT
View Forum Message <> Reply to Message

"Karl Schultz" <kschultz@researchsystems.com> wrote in message
news:9gvt3o$lkk$1@news.rsinc.com...
> The order in which the polygons are drawn is extremely important when
using
> alpha blending.
>
> In the first case:
>
> 1) The red orb is drawn first without alpha blending. So far, so good.
>
> 2) The green orb is then drawn with alpha blending. You don't know
> beforehand

> which polygons in the green orb draw first. What is happening is that the
> top
> (pos z) part of the orb is drawing first. This is the transparent part,
and
> after we draw
> this part, we see the red orb and parts of the background, as we expect.
>

So the order in which the polygon VERTICIES are drawn per the polygon connectivity matters as well as order of the objects in the view? So in the first case, if the polygon connectivity array was "flipped" so that the orb polys would be drawn back to front the image would render correctly. I would test this but it looks like the orb is comprised of a top and bottom drawn from triangles and a body from rectangles so I can't easily flip the polygon array. (this also explains the green and red pimples you see with my example.)

A single polygon that is transparent on opposing sides can not be rendered correctly. In order to render this correctly you would have to break it up into parts and order them correctly in the view according to the camera position in z.

If I understand you then this all makes sense. I never thought about it and assumed that as long as objects were ordered correctly in the view then they would be drawn correctly (which is true until you start alpha blending).
This actually sheds some light on a number of subtle issues we have been having with alpha blending.

Thanks.

-Rick

> Now drawing these transparent
> polygons also sets the Z buffer in this part of the screen. So, as we
draw
> the rest of
> the green orb, the bottom or -Z part, the Z buffer test fails and these
> polys do not
> draw. The part of the screen where the transparent polys are does not
> change, and
> you end up seeing background and red orb where you instead
> wanted to see the inside of the green orb.
>
> In the second case:
>

> - The transparent part of the green orb is drawn last, because you moved
it
> from
> +z to -z, in terms of the orb coords. (Assuming that you've spun it
around
> 180 in xobjview)
> So, the side opposite the transparent "hole" draws first since it is in
the
> +z
> part of the orb. This parts makes it onto the screen and then the
> transparent polygons are drawn on top, letting you see the inside.
>
> Hope this helps. The general point is that you really have to be very
> careful in
> layering primitives when drawing with alpha. The layering may need to
take
> into account the viewing angle and the geometry of the primitives.
>
> For example, in this case, you may have to rotate the orb so that it
always
> draws
> back-to-front and move the transparent part so that it is always "in
front",
> with
> respect to the viewing direction.
>

> Karl
>
> "Rick Towler" <rtowler@u.washington.edu> wrote in message
> news:9gtvnl$q08$1@nntp6.u.washington.edu...
>> I have been experimenting using the alpha channel to represent
confidence
> in
>> a data set. I produce a polygon object representing the data and then
>> texture map the polygon accordingly. But, I have run into an issue that
I
>> can't resolve.
>>
>> I have attached a simple example (requires at least IDL 5 something and
24
>> bit display). I create 2 orb objects, one green (the olive) and one red
>> (the pimento).
>>
>> At first I place the olive in front of the pimento, then I make the near
>> surface of the olive transparent. Rendering in hardware for the most

part
>> works except between the last opaque vertices and the first transparent
>> vertices I get a white (background color) ring where you can actually
see
>> thru the entire olive to the pimento behind it. If your imagination is
>> failing I'll remind you that we are just supposed to be looking into the
>> inside of the olive, we shouldn't actually see the pimento thru the
olive.
>> Rendering in software, things get even more wacky. I don't see the
inside
>> of my olive at all. Xray vision I guess. What is interesting to note
is
>> that this ring I get when rendering in hardware renders just like the
all
> of
>> the transparent vertices in software.
>>
>>
>> I then flip the pimento and the hole in my olive. The pimento now is in
>> front of the olive and the "hole" is created on the back side of the
green
>> orb. Now when I rotate the olive around so I can see in my hole, things
>> look pretty good. The astute observer will notice that the ring is
still
>> there (you will see a little red pimple if viewed from the right angle)
> but
>> otherwise things look good. In software mode, things seem to work.
>>
>>
>> What is going on in the first case? Why does software rendering fail to
>> render the inside of the olive? What am I missing?
>>
>> I would be happy to provide images upon request.
>>
>> -Rick Towler
>
>
>

Subject: Re: texture map irregularities OR pimento problems
Posted by Karl Schultz on Fri, 22 Jun 2001 23:11:01 GMT

View Forum Message <> Reply to Message

"Rick Towler" <rtowler@u.washington.edu> wrote in message
news:UZNY6.226424$p33.4540384@news1.sttls1.wa.home.com...
>
> "Karl Schultz" <kschultz@researchsystems.com> wrote in message

> news:9gvt3o$lkk$1@news.rsinc.com...
>> The order in which the polygons are drawn is extremely important when
> using
>> alpha blending.
>>
>> In the first case:
>>
>> 1) The red orb is drawn first without alpha blending. So far, so good.
>>
>> 2) The green orb is then drawn with alpha blending. You don't know
>> beforehand
>> which polygons in the green orb draw first. What is happening is that
the
>> top
>> (pos z) part of the orb is drawing first. This is the transparent part,
> and
>> after we draw
>> this part, we see the red orb and parts of the background, as we expect.
>>
>
> So the order in which the polygon VERTICIES are drawn per the polygon
> connectivity matters as well as order of the objects in the view?

Yes, but it is really more the order in which the POLYGONS (in the mesh generated by Orb) are drawn according to the connectivity list. (Many vertices are shared between polygons, so it is really the polygons we are worried about, but I think we're saying the same thing) And the drawing order really only matters if you are using alpha.

> So in the
> first case, if the polygon connectivity array was "flipped" so that the
orb
> polys would be drawn back to front the image would render correctly.

Yes.

> I
> would test this but it looks like the orb is comprised of a top and bottom
> drawn from triangles and a body from rectangles so I can't easily flip the
> polygon array. (this also explains the green and red pimples you see with
my
> example.)

We ship the source code for Orb, so you can see what is what is going on if you want. But I wouldn't code anything that relied on an orb internal. (OO programming)

But I think that you could also "flip" any arbitrary polygon connectivity

list pretty easily without knowing anything about its shape or what created
it. Just walk the connectivity list and copy each polygon descriptor to a
new descriptor list where you append each descriptor to the front of the new
list.

A descriptor looks like: (list of LONG)
n v0 v1 v2 ... vn-1

where n is the number of verts in that poly.

Anyway, in this case, it might be better to just rotate the orb with a
grModel. Luckily, orbs look the same (disregarding tesselation details)
after a rotation, so you could just rotate it so that the top (+z) is
pointing away from the viewer. Since we know (by observation) that the orb
draws from + to - z, that will make the back part of the orb draw first.

In your first example, applying a 180 degree rotation about X or Y would do
the trick.

This still relies on Orb internals - Orb could change in such a way that
this assumption on drawing order would no longer be valid. To be safe, you'd
have to control the generation of the mesh itself.

> A single polygon that is transparent on opposing sides can not be rendered
> correctly. In order to render this correctly you would have to break it
up
> into parts and order them correctly in the view according to the camera
> position in z.

Do you mean a single polygon *mesh*? If so, right. Unless you can rotate
the object so that the drawing order of the polygons in the mesh gives you
the ordering you want. For more general and more complex objects than a
sphere, this could be hard or impossible. In the end, you'd have to control
the order somehow, and breaking it up would work.

> If I understand you then this all makes sense. I never thought about it
and
> assumed that as long as objects were ordered correctly in the view then
they
> would be drawn correctly (which is true until you start alpha blending).
> This actually sheds some light on a number of subtle issues we have been
> having with alpha blending.

Right. I think of this sort of thing as more of "compositing", instead of
just drawing z-buffered polygons. You have to control your scene much more
carefully.