Subject: Re: texture_coord
Posted by Karl Schultz on Fri, 02 Nov 2001 16:39:18 GMT
View Forum Message <> Reply to Message

"David Fanning" <david@dfanning.com> wrote in message
news:MPG.164b980a9deb6a55989744@news.frii.com...
> Note that the article talks about a couple of unresolved
> issues. First, when I position the image as above, I don't
> seem to have control over what color the *rest* of the
> surface is.

Yes, that's a little hard in this context. In fact, my difficulty in explaining this may suggest that we need to add something to IDL to make this easier. Some discussion may help.

At least with the code posted in this thread, we were just letting the "unused" texture coords stay (0,0), which means that the color on the rest of the surface would take on the color of the (0,0) texel, whatever that is. See next topic.

> Second, the positioned image seems to have
> problems around its edges. I suspect both of these problems
> may be related, but so far I have made no progress resolving
> them. I'm open to any and all ideas.

Right. I noticed this too. There's some interpolation going on along the edge of the surface. One thing to keep in mind is that there isn't always an one-to-one relationship between texels and pixels. It may take several texels to decide what color to make a pixel, as is the case where the texture has a higher sampling than the screen. Or, a single texel may be used to determine the color of many pixels in the opposite case. You might be seeing texel (0,0) and the texel from an edge of the texture being combined to determine the color of a pixel along the edge. This can lead to somewhat random-looking results.

In fact, the texel interpolation is a whole lot worse than this. Suppose that you are mapping a texture onto a sub-surface with corners [20,20] and [30,30]. The texture coordinate at [20,20] is [0,0]. The texture coordinate at [20,19] is also [0,0], so there is no real problem there. But if you look at the texture coordinate at [28,20] and at [28,19] we see that two adjacent texture coords are something like [0.9,0.0] and [0.0,0.0]. This is really bad because OpenGL will take all the texels (from the image) between (normallized) [0.9,0.0] and [0,0], average them together, and then use that color value as one of the colors used to decide the color of the pixel in that area. It gets a lot worse if you go over to [30,30] where we would average all the image texels in a diagonal line across the image. Yuk. OpenGL is doing what we tell it to, but not what we want.

OpenGL has a LOT of facilities in its texture mapping support to control border issues, which indicates to me that it is not a simple problem. IDL doesn't expose all these controls.

One step in attacking the problem is to pre-process your image to put a border around it. Make the color whatever you'd like the "rest" of the surface to look like. And you might try it with one-pixel borders, and perhaps two or three. I extended your program (texture_surface) to do this and I got a nice black background since I made my borders black.

But this didn't completely solve the problem. The left and the bottom borders look ok - black. But the top and right edges have smudged up colors where the black border should be. This is caused by the texel interpolation across the image I mentioned above. How to fix this? More work. The texture coords of the vertices ADJACENT to the area where the texture is mapped need to be something other than [0,0]. Using the above example again, the texture coordinates at [28,20] should be [0.9,0.0] and the texture coordinates at [28,19] need to be [0.9,0.0] as well. This will cause the texels accessed from the image to be the same (the new border texels in this case) and we should get black. In fact, it may make sense to set the texture coords at [28,0:19] to [0.9,0.0]. This avoids the [0.9,0.0]->[0.0,0.0] texel interpolation across the image.

I also tried this with your program and got pretty encouraging results, although I didn't implement a full, general solution. Maybe I'll work on it.

The bottom-line is that we were getting lazy by not setting the texture coordinates of the vertices of the "rest" of the surface to reasonable values. This caused a major discontinuity in the texture interpolation.

>
> Oh, by the way, I think I was wrong about the resolution
> of the surface. Making the surface bigger does not seem
> to affect the resolution of the image on the surface
> at all.

I almost posted about this topic yesterday. Right, the number of polygons (facets) in the surface won't have an effect on the appearance of the image. The texture image is interpolated across each facet, using the texture coords of the facet to determine what part of the image is used. If you generate more facets, you will have smaller steps in the texture coordinates across the facets and you end up with the same thing. If you wanted to do something other than a linear texture mapping, like some sort of morphing, then you might want more facets to give you more control.

You may see some difference in a more geometric sense. For example, if you have an implicit surface (generated by some function) that is pretty curvy,

you'll get a better looking and more accurate surface as you increase the number of facets, texture or no texture. For simpler surfaces, fewer facets suffice, textured or not. People often map textures onto 4-vertex planar polygons or surfaces so that they can display an image in a more flexible way. You can't really manipluate an IDLgrImage very well with all the model transforms, so if you wanted to display an image with an arbitrary transform, you can map it onto a simple polygon and orient it however you want. You also gain a lot of functionality in the areas of stretching and transparency. Anyway, one facet is enough if all you want is a flat surface. The image texels are interpolated across the single facet.

---