Subject: Re: IDLgrVolume RENDER_STEP does not scale
Posted by Karl Schultz on Tue, 22 Oct 2002 16:29:10 GMT
View Forum Message <> Reply to Message

"Sebastian Loebbert" <sebaaihb@peach.zrz.TU-Berlin.DE> wrote in message
news:Pine.LNX.4.44.0210211349220.20769-100000@peach.zrz.TU-B erlin.DE...
> Hi all,
>
> I am trying to implement a Level-Of-Detail system for a volume renderer:
> While the volume is rotated, only a low-detailed rendering shall be done
> (in the dvolrendr.pro example they only show a wire frame/ coord system).
> I tried to use RENDER_STEP to get high speed but low quality renderings,
> but I can't improve rendering time for more than a factor of about 5 for a
> given dataset no matter what I use for RENDER_STEP:
> E.g. for a 128^3 volume in a 300^2 window i get the following
> RENDER_STEP vs time values:
> [1,1,1] - 4.9 s
> [1,1,3] - 2.6 s
> [1,1,10] - 1,65 s
> [1,1,15] - 1.6 s
> [10,10,10] - 1.2 s
> for reference: merging two 300^2 rgb color images and drawing the
> resulting takes 0.12 s
>
> Theoretically, I would expect a factor 1000 between [10,10,10] and [1,1,1]
> plus some overhead, why is the difference so small?

I'll try to take a look at this.

I suggest that you also take advantage of our excellent RSI Tech Support group for reporting problems like these. They've got very high customer satisfaction numbers that I believe are well above the industry average. If you check in with them about your problem, you can track the issue more closely.

> Is there another method for getting high-speed/low-quality images?

There is, and it is pretty compilicated. I'll try to outline the general approach here. I do have some code that does something like this, but I'm not sure how "open" it is.

The basic idea is that you build a series of stacked translucent texture-mapped polygons and then render the polygons. Most GL implementations are optimized for this type of rendering, so the result is pretty fast. I can interactively rotate a volume that is about the same size as your volume. The visual differences are pretty subtle and are close enough, in my opinion, to use this form for interactive volume rendering.

The basic approach is:

1) Figure out how many texture maps and polygons you need. Decide on the direction that you will view the volume and figure out which dimension of the volume is most perpendicular to the viewing vector. The "slices" are the 2D volume planes most perpendicular to this vector. You'll need a texture map and polygon for each slice.
2) Create a RGBA texture map (IDLgrImage) for each slice. If you have a simple one-byte-per-sample sort of volume with no color or opacity tables, just replicate the sample value in all four channels.
3) Create a polygon for each slice. Its geometry covers the extent of a slice and of course uses the associated texture map.
4) Create a model and put the polygon slices in it so that it renders back-to-front.

Improvements:
- If your volume dimensions are not powers of 2, IDL will stretch your texture map to the next higher power of two, which could introduce unwanted resampling artifacts. To combat this, make your IDLgrImage dimensions powers of two and copy your texture data into a subset of this image. Then use texture coordinates to map the used subset of the texture onto the polygon.

- If you intend to rotate this "volume" with a trackball or something, you'll want a to make 6 such stacks of polygons and select the appropriate one to draw, based on the axis and direction.

That's about it. We can discuss this more if there is interest.

Karl
