John Carmack Archive - .plan (2004)

http://www.team5150.com/~andrew/carmack

March 18, 2007

Contents

Chapter 1

December

1.1 Welcome (Dec 31, 2004)

I get a pretty steady trickle of emails from people hoping for .plan file updates. There were two main factors involved in my not doing updates for a long time - a good chunk of my time and interest was sucked into Armadillo Aerospace, and the fact that the work I had been doing at Id for the last half of Doom 3 development was basically pretty damn boring.

The Armadillo work has been very rewarding from a learning-lots-of-new-stuff perspective, and I'm still committed to the vehicle development, even post X-Prize, but the work at Id is back to a high level of interest now that we are working on a new game with new technology. I keep running across topics that are interesting to talk about, and the Armadillo updates have been a pretty good way for me to organize my thoughts, so I'm going to give it a more general try here. .plan files were appropriate ten years ago, and sort of retro-cute several years ago, but I'll be sensible and use the web.

I'm not quite sure what the tone is going to be - there will probably be some general interest stuff, but a bunch of things will only be of interest to hardcore graphics geeks.

I have had some hesitation about doing this because there are a hundred times as many people interested in listening to me talk about games / graphics / computers as there are people interested in rocket fabrication, and my mailbox is already rather time consuming to get through.

If you really, really want to email me, add a "[JC]" in the subject header so the mail gets filtered to a mailbox that isn't clogged with spam. I can't respond to most of the email I get, but I do read everything that doesn't immediately scan as spam. Unfortunately, the probability of getting an answer from me doesn't have a lot of correlation with the quality of the question, because what I am doing at the instant I read it is more dominant, and there is even a negative correlation for "deep" questions that I don't want to make an off-the-cuff response to.

Quake 3 Source

I intended to release the Q3 source under the GPL by the end of 2004, but we had another large technology licensing deal go through, and it would be poor form to make the source public a few months after a company paid hundreds of thousands of dollars for full rights to it. True, being public under the GPL isn't the same as having a royalty free license without the need to disclose the source, but I'm pretty sure there would be some hard feelings.

Previous source code releases were held up until the last commercial license of the technology shipped, but with the evolving nature of game engines today, it is a lot less clear. There are still bits of early Quake code in Half Life 2, and the remaining licensees of Q3 technology intend to continue their internal developments along similar lines, so there probably won't be nearly as sharp a cutoff as before. I am still committed to making as much source public as I can, and I won't wait until the titles from the latest deal have actually shipped, but it is still going to be a little while before I feel comfortable doing the release.

Random Graphics Thoughts

Years ago, when I first heard about the inclusion of derivative instructions in fragment programs, I couldn't think of anything off hand that I wanted them for. As I start working on a new generation of rendering code, uses for them come up a lot more often than I expected.

I can't actually use them in our production code because it is an Nvidia-only feature at the moment, but it is convenient to do experimental code with the nv_fragment_program extension before figuring out various ways to build funny texture mip maps so that the built in texture filtering hardware calculates a value somewhat like the derivative I wanted.

If you are basically just looking for plane information, as you would for modifying things with texture magnification or stretching shadow buffer filter kernels, the derivatives work out pretty well. However, if you are looking at a derived value, like a normal read from a texture, the results are almost useless because of the way they are calculated. In an ideal world, all of the samples to be differenced would be calculated at once, then the derivatives calculated from there, but the hardware only calculates 2x2 blocks at a time. Each of the four pixels in the block is given the same derivative, and there is no influence from neighboring pixels. This gives derivative information that is basically half the resolution of the screen and sort of point sampled. You can often see this effect with bump mapped environment mapping into a mip-mapped cube map, where the texture LOD changes discretely along the 2x2 blocks. Explicitly coloring based on the derivatives of a normal map really shows how nasty the calculated value is.

Speaking of bump mapped environment sampling.. I spent a little while tracking down a highlight that I thought was misplaced. In retrospect it is obvious, but I never considered the artifact before: With a bump mapped surface, some of the on-screen normals will actually be facing away from the viewer. This causes minor problems with lighting, but when you are making a reflection vector from it, the vector starts reflecting into the opposite hemisphere, resulting in some sky-looking pixels near bottom edges on the model. Clamping the surface normal to not face away isn't a good solution, because you get areas that "see right through" to the environment map, because a reflection past a clamped perpendicular vector doesn't change the viewing vector. I could probably ramp things based on the geometric normal somewhat, and possibly pre-calculate some data into the normal maps, but I decided it wasn't a significant enough issue to be worth any more development effort or speed hit.

Speaking of cube maps.. The edge filtering on cube maps is showing up as an issue for some algorithms. The hardware basically picks a face, then treats it just like a 2D texture. This is fine in the middle of the texture, but at the edges (which are a larger and larger fraction as size decreases) the filter kernel just clamps instead of being able to sample the neighbors in an adjacent cube face. This is generally a non-issue for classic environment mapping, but when you start using cube map lookups with explicit LOD bias inputs (say, to simulate variable specular powers into an environment map) you can wind up with a surface covered with six constant color patches instead of the smoothly filtered coloration you want. The classic solution would be to implement border texels, but that is pretty nasty for the hardware and API, and would require either the application or the driver to actually copy the border texels from all the other faces. Last I heard, upcoming hardware was going to start actually fetching from the other side textures directly. A second-tier chip company claimed to do this correctly a while ago, but I never actually tested it.

Topics continue to chain together, I'll probably write some more next week.