John Carmack Archive - .plan (2002)

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

March 18, 2007

Contents

Chapter 1

February

1.1 Last month I wrote the Radeon 8500 support for Doom. (Feb 11, 2002)

The bottom line is that it will be a fine card for the game, but the details are sort of interesting.

I had a pre-production board before Siggraph last year, and we were discussing the possibility of letting ATI show a Doom demo behind closed doors on it. We were all very busy at the time, but I took a shot at bringing up support over a weekend. I hadn't coded any of the support for the custom ATI extensions yet, but I ran the game using only standard OpenGL calls (this is not a supported path, because without bump mapping everything looks horrible) to see how it would do. It didn't even draw the console correctly, because they had driver bugs with texGen. I thought the odds were very long against having all the new, untested extensions working properly, so I pushed off working on it until they had revved the drivers a few more times.

My judgment was colored by the experience of bringing up Doom on the original Radeon card a year earlier, which involved chasing a lot of driver bugs. Note that ATI was very responsive, working closely with me on it, and we were able to get everything resolved, but I still had no expectation

that things would work correctly the first time.

Nvidia's OpenGL drivers are my "gold standard", and it has been quite a while since I have had to report a problem to them, and even their brand new extensions work as documented the first time I try them. When I have a problem on an Nvidia, I assume that it is my fault. With anyone else's drivers, I assume it is their fault. This has turned out correct almost all the time. I have heard more anecdotal reports of instability on some systems with Nivida drivers recently, but I track stability separately from correctness, because it can be influenced by so many outside factors.

ATI had been patiently pestering me about support for a few months, so last month I finally took another stab at it. The standard OpenGL path worked flawlessly, so I set about taking advantage of all the 8500 specific features. As expected, I did run into more driver bugs, but ATI got me fixes rapidly, and we soon had everything working properly. It is interesting to contrast the Nvidia and ATI functionality:

The vertex program extensions provide almost the same functionality. The ATI hardware is a little bit more capable, but not in any way that I care about. The ATI extension interface is massively more painful to use than the text parsing interface from nvidia. On the plus side, the ATI vertex programs are invariant with the normal OpenGL vertex processing, which allowed me to reuse a bunch of code. The Nvidia vertex programs can't be used in multipass algorithms with standard OpenGL passes, because they generate tiny differences in depth values, forcing you to implement EVERYTHING with vertex programs. Nvidia is planning on making this optional in the future, at a slight speed cost.

I have mixed feelings about the vertex object / vertex array range extensions. ATI's extension seems more "right" in that it automatically handles synchronization by default, and could be implemented as a wire protocol, but there are advantages to the VAR extension being simply a hint. It is easy to have a VAR program just fall back to normal virtual memory by not setting the hint and using malloc, but ATI's extension requires different function calls for using vertex objects and normal vertex arrays.

The fragment level processing is clearly way better on the 8500 than on the Nvidia products, including the latest GF4. You have six individual textures, but you can access the textures twice, giving up to eleven pos-

sible texture accesses in a single pass, and the dependent texture operation is much more sensible. This wound up being a perfect fit for Doom, because the standard path could be implemented with six unique textures, but required one texture (a normalization cube map) to be accessed twice. The vast majority of Doom light / surface interaction rendering will be a single pass on the 8500, in contrast to two or three passes, depending on the number of color components in a light, for GF3/GF4 (*note GF4 bitching later on).

Initial performance testing was interesting. I set up three extreme cases to exercise different characteristics:

A test of the non-textured stencil shadow speed showed a GF3 about 20% faster than the 8500. I believe that Nvidia has a slightly higher performance memory architecture.

A test of light interaction speed initially had the 8500 significantly slower than the GF3, which was shocking due to the difference in pass count. ATI identified some driver issues, and the speed came around so that the 8500 was faster in all combinations of texture attributes, in some cases 30+% more. This was about what I expected, given the large savings in memory traffic by doing everything in a single pass.

A high polygon count scene that was more representative of real game graphics under heavy load gave a surprising result. I was expecting ATI to clobber Nvidia here due to the much lower triangle count and MUCH lower state change functional overhead from the single pass interaction rendering, but they came out slower. ATI has identified an issue that is likely causing the unexpected performance, but it may not be something that can be worked around on current hardware.

I can set up scenes and parameters where either card can win, but I think that current Nvidia cards are still a somewhat safer bet for consistent performance and quality.

On the topic of current Nvidia cards:

Do not buy a GeForce4-MX for Doom.

Nvidia has really made a mess of the naming conventions here. I always thought it was bad enough that GF2 was just a speed bumped GF1, while

GF3 had significant architectural improvements over GF2. I expected GF4 to be the speed bumped GF3, but calling the NV17 GF4-MX really sucks.

GF4-MX will still run Doom properly, but it will be using the NV10 codepath with only two texture units and no vertex shaders. A GF3 or 8500 will be much better performers. The GF4-MX may still be the card of choice for many people depending on pricing, especially considering that many games won't use four textures and vertex programs, but damn, I wish they had named it something else.

As usual, there will be better cards available from both Nvidia and ATI by the time we ship the game.

8:50 pm addendum: Mark Kilgard at Nvidia said that the current drivers already support the vertex program option to be invarint with the fixed function path, and that it turned out to be one instruction FASTER, not slower.

Chapter 2

March

2.1 Mar 15, 2002

Mark Kilgard and Cass Everitt at Nvidia have released a paper on shadow volume rendering with several interesting bits in it. They also include a small document that I wrote a couple years ago about my discovery process during the development of some of the early Doom technology.

http://developer.nvidia.com/view.asp?IO=robust_shadow_volumes

Chapter 3

June

3.1 The Matrox Parhelia Report (Jun 25, 2002)

The executive summary is that the Parhelia will run Doom, but it is not performance competitive with Nvidia or ATI.

Driver issue remain, so it is not perfect yet, but I am confident that Matrox will resolve them.

The performance was really disappointing for the first 256 bit DDR card. I tried to set up a ”poster child” case that would stress the memory subsystem above and beyond any driver or triangle level inefficiencies, but I was unable to get it to ever approach the performance of a GF4.

The basic hardware support is good, with fragment flexibility better than GF4 (but not as good as ATI 8500), but it just doesn't keep up in raw performance. With a die shrink, this chip could probably be a contender, but there are probably going to be other chips out by then that will completely eclipse this generation of products.

None of the special features will be really useful for Doom:

The 10 bit color framebuffer is nice, but Doom needs more than 2 bits of destination alpha when a card only has four texture units, so we can't use it.

Anti aliasing features are nice, but it isn't all that fast in minimum feature mode, so nobody is going to be turning on AA. The same goes for "surround gaming". While the framerate wouldn't be 1/3 the base, it would still probably be cut in half.

Displacement mapping. Sigh. I am disappointed that the industry is still pursuing any quad based approaches. Haven't we learned from the stellar success of 3DO, Saturn, and NV1 that quads really suck? In any case, we can't use any geometry amplification scheme (including ATI's truform) in conjunction with stencil shadow volumes.

3.2 More graphics card notes (Jun 27, 2002)

I need to apologize to Matrox - their implementation of hardware displacement mapping is NOT quad based. I was thinking about a certain other companies proposed approach. Matrox's implementation actually looks quite good, so even if we don't use it because of the geometry amplification issues, I think it will serve the noble purpose of killing dead any proposal to implement a quad based solution.

I got a 3Dlabs P10 card in last week, and yesterday I put it through its paces. Because my time is fairly over committed, first impressions often determine how much work I devote to a given card. I didn't speak to ATI for months after they gave me a beta 8500 board last year with drivers that rendered the console incorrectly. :)

I was duly impressed when the P10 just popped right up with full functional support for both the fallback ARB_ extension path (without specular highlights), and the NV10 NVidia register combiners path. I only saw two issues that were at all incorrect in any of our data, and one of them is debatable. They don't support NV_vertex_program_1_1, which I use for the NV20 path, and when I hacked my programs back to 1.0 support for testing, an issue did show up, but still, this is the best showing from a new board from any company other than Nvidia.

It is too early to tell what the performance is going to be like, because they don't yet support a vertex object extension, so the CPU is hand feeding all

the vertex data to the card at the moment. It was faster than I expected for those circumstances.

Given the good first impression, I was willing to go ahead and write a new back end that would let the card do the entire Doom interaction rendering in a single pass. The most expedient sounding option was to just use the Nvidia extensions that they implement, NV_vertex_program and NV_register_combiners, with seven texture units instead of the four available on GF3/GF4. Instead, I decided to try using the prototype OpenGL 2.0 extensions they provide.

The implementation went very smoothly, but I did run into the limits of their current prototype compiler before the full feature set could be implemented. I like it a lot. I am really looking forward to doing research work with this programming model after the compiler matures a bit. While the shading languages are the most critical aspects, and can be broken out as extensions to current OpenGL, there are a lot of other subtle-but-important things that are addressed in the full OpenGL 2.0 proposal.

I am now committed to supporting an OpenGL 2.0 renderer for Doom through all the spec evolutions. If anything, I have been somewhat remiss in not pushing the issues as hard as I could with all the vendors. Now really is the critical time to start nailing things down, and the decisions may stay with us for ten years.

A GL2 driver won't give any theoretical advantage over the current back ends optimized for cards with 7+ texture capability, but future research work will almost certainly be moving away from the lower level coding practices, and if some new vendor pops up (say, Rendition back from the dead) with a next-gen card, I would strongly urge them to implement GL2 instead of proprietary extensions.

I have not done a detailed comparison with Cg. There are a half dozen C-like graphics languages floating around, and honestly, I don't think there is a hell of a lot of usability difference between them at the syntax level. They are all a whole lot better than the current interfaces we are using, so I hope syntax quibbles don't get too religious. It won't be too long before all real work is done in one of these, and developers that stick with the lower level interfaces will be regarded like people that write all-assembly

PC applications today. (I get some amusement from the all-assembly crowd, and it can be impressive, but it is certainly not effective)

I do need to get up on a soapbox for a long discourse about why the upcoming high level languages MUST NOT have fixed, queried resource limits if they are going to reach their full potential. I will go into a lot of detail when I get a chance, but drivers must have the right and responsibility to multipass arbitrarily complex inputs to hardware with smaller limits. Get over it.