John Carmack Archive - .plan (2001)

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

March 18, 2007

Contents

Chapter 1

February

1.1 GeForce 3 Overview (Feb 22, 2001)

I just got back from Tokyo, where I demonstrated our new engine running under MacOS-X with a GeForce 3 card. We had quite a bit of discussion about whether we should be showing anything at all, considering how far away we are from having a title on the shelves, so we probably aren't going to be showing it anywhere else for quite a while.

We do run a bit better on a high end wintel system, but the Apple performance is still quite good, especially considering the short amount of time that the drivers had before the event.

It is still our intention to have a simultaneous release of the next product on Windows, MacOS-X, and Linux.

Here is a dump on the GeForce 3 that I have been seriously working with for a few weeks now:

The short answer is that the GeForce 3 is fantastic. I haven't had such an impression of raising the performance bar since the Voodoo 2 came out, and there are a ton of new features for programmers to play with.

Graphics programmers should run out and get one at the earliest possible time. For consumers, it will be a tougher call. There aren't any appli-

cations our right now that take proper advantage of it, but you should still be quite a bit faster at everything than GF2, especially with anti-aliasing. Balance that against whatever the price turns out to be.

While the Radeon is a good effort in many ways, it has enough shortfalls that I still generally call the GeForce 2 ultra the best card you can buy right now, so Nvidia is basically dethroning their own product.

It is somewhat unfortunate that it is labeled GeForce 3, because GeForce 2 was just a speed bump of GeForce, while GF3 is a major architectural change. I wish they had called the GF2 something else.

The things that are good about it:

Lots of values have additional internal precision, like texture coordinates and rasterization coordinates. There are only a few places where this matters, but it is nice to be cleaning up. Rasterization precision is about the last thing that the multi-thousand dollar workstation boards still do any better than the consumer cards.

Adding more texture units and more register combiners is an obvious evolutionary step.

An interesting technical aside: when I first changed something I was doing with five single or dual texture passes on a GF to something that only took two quad texture passes on a GF3, I got a surprisingly modest speedup. It turned out that the texture filtering and bandwidth was the dominant factor, not the frame buffer traffic that was saved with more texture units. When I turned off anisotropic filtering and used compressed textures, the GF3 version became twice as fast.

The 8x anisotropic filtering looks really nice, but it has a 30%+ speed cost. For existing games where you have speed to burn, it is probably a nice thing to force on, but it is a bit much for me to enable on the current project. Radeon supports 16x aniso at a smaller speed cost, but not in conjunction with trilinear, and something is broken in the chip that makes the filtering jump around with triangular rasterization dependencies.

The depth buffer optimizations are similar to what the Radeon provides, giving almost everything some measure of speedup, and larger ones avail-

able in some cases with some redesign.

3D textures are implemented with the full, complete generality. Radeon offers 3D textures, but without mip mapping and in a non-orthogonal manner (taking up two texture units).

Vertex programs are probably the most radical new feature, and, unlike most "radical new features", actually turn out to be pretty damn good. The instruction language is clear and obvious, with wonderful features like free arbitrary swizzle and negate on each operand, and the obvious things you want for graphics like dot product instructions.

The vertex program instructions are what SSE should have been.

A complex setup for a four-texture rendering pass is way easier to understand with a vertex program than with a ton of texgen/texture matrix calls, and it lets you do things that you just couldn't do hardware accelerated at all before. Changing the model from fixed function data like normals, colors, and texcoords to generalized attributes is very important for future progress.

Here, I think Microsoft and DX8 are providing a very good benefit by forcing a single vertex program interface down all the hardware vendor's throats.

This one is truly stunning: the drivers just worked for all the new features that I tried. I have tested a lot of pre-production 3D cards, and it has never been this smooth.

The things that are indifferent:

I'm still not a big believer in hardware accelerated curve tessellation. I'm not going to go over all the reasons again, but I would have rather seen the features left off and ended up with a cheaper part.

The shadow map support is good to get in, but I am still unconvinced that a fully general engine can be produced with acceptable quality using shadow maps for point lights. I spent a while working with shadow buffers last year, and I couldn't get satisfactory results. I will revisit that work now that I have GeForce 3 cards, and directly compare it with my current approach.

At high triangle rates, the index bandwidth can get to be a significant thing. Other cards that allow static index buffers as well as static vertex buffers will have situations where they provide higher application speed. Still, we do get great throughput on the GF3 using vertex array range and glDrawElements.

The things that are bad about it:

Vertex programs aren't invariant with the fixed function geometry paths. That means that you can't mix vertex program passes with normal passes in a multipass algorithm. This is annoying, and shouldn't have happened.

Now we come to the pixel shaders, where I have the most serious issues. I can just ignore this most of the time, but the way the pixel shader functionality turned out is painfully limited, and not what it should have been.

DX8 tries to pretend that pixel shaders live on hardware that is a lot more general than the reality.

Nvidia's OpenGL extensions expose things much more the way they actually are: the existing register combiners functionality extended to eight stages with a couple tweaks, and the texture lookup engine is configurable to interact between textures in a list of specific ways.

I'm sure it started out as a better design, but it apparently got cut and cut until it really looks like the old BumpEnvMap feature writ large: it does a few specific special effects that were deemed important, at the expense of a properly general solution.

Yes, it does full bumpy cubic environment mapping, but you still can't just do some math ops and look the result up in a texture. I was disappointed on this count with the Radeon as well, which was just slightly too hardwired to the DX BumpEnvMap capabilities to allow more general dependent texture use.

Enshrining the capabilities of this mess in DX8 sucks. Other companies had potentially better approaches, but they are now forced to dumb them down to the level of the GF3 for the sake of compatibility. Hopefully we can still see some of the extra flexibility in OpenGL extensions.

The future:

I think things are going to really clean up in the next couple years. All of my advocacy is focused on making sure that there will be a completely clean and flexible interface for me to target in the engine after DOOM, and I think it is going to happen.

The market may have shrunk to just ATI and Nvidia as significant players. Matrox, 3D labs, or one of the dormant companies may surprise us all, but the pace is pretty frantic.

I think I would be a little more comfortable if there was a third major player competing, but I can't fault Nvidia's path to success.

Chapter 2

November

2.1 Nov 16, 2001

Driver optimizations have been discussed a lot lately because of the quake3 name checking in ATI's recent drivers, so I am going to lay out my position on the subject.

There are many driver optimizations that are pure improvements in all cases, with no negative effects. The difficult decisions come up when it comes to "trades" of various kinds, where a change will give an increase in performance, but at a cost.

Relative performance trades. Part of being a driver writer is being able to say "I don't care if stippled, anti-aliased points with texturing go slow", and optimizing accordingly. Some hardware features, like caches and hierarchical buffers, may be advantages on some apps, and disadvantages on others. Command buffer sizes often tune differently for different applications.

Quality trades. There is a small amount of wiggle room in the specs for pixel level variability, and some performance gains can be had by leaning towards the minimums. Most quality trades would actually be conformance trades, because the results are not exactly conformant, but they still do "roughly" the right thing from a visual standpoint. Compressing

textures automatically, avoiding blending of very faint transparent pixels, using a 16 bit depth buffer, etc. A good application will allow the user to make most of these choices directly, but there is good call for having driver preference panels to enable these types of changes on naive applications. Many drivers now allow you to quality trade in an opposite manner - slowing application performance by turning on anti-aliasing or anisotropic texture filtering.

Conformance trades. Most conformance trades that happen with drivers are unintentional, where the slower, more general fallback case just didn't get called when it was supposed to, because the driver didn't check for a certain combination to exit some specially optimized path. However, there are optimizations that can give performance improvements in ways that make it impossible to remain conformant. For example, a driver could choose to skip storing of a color value before it is passed on to the hardware, which would save a few cycles, but make it impossible to correctly answer glGetFloatv(GL_CURRENT_COLOR, buffer).

Normally, driver writers will just pick their priorities and make the trades, but sometimes there will be a desire to make different trades in different circumstances, so as to get the best of both worlds.

Explicit application hints are a nice way to offer different performance characteristics, but that requires cooperation from the application, so it doesn't help in an ongoing benchmark battle. OpenGL's glHint() call is the right thought, but not really set up as flexibly as you would like. Explicit extensions are probably the right way to expose performance trades, but it isn't clear to me that any conformant trade will be a big enough difference to add code for.

End-user selectable optimizations. Put a selection option in the driver properties window to allow the user to choose which application class they would like to be favored in some way. This has been done many times, and is a reasonable way to do things. Most users would never touch the setting, so some applications may be slightly faster or slower than in their "optimal benchmark mode".

Attempt to guess the application from app names, window strings, etc. Drivers are sometimes forced to do this to work around bugs in established software, and occasionally they will try to use this as a cue for cer-

tain optimizations.

My positions:

Making any automatic optimization based on a benchmark name is wrong. It subverts the purpose of benchmarking, which is to gauge how a similar class of applications will perform on a tested configuration, not just how the single application chosen as representative performs.

It is never acceptable to have the driver automatically make a conformance tradeoff, even if they are positive that it won't make any difference. The reason is that applications evolve, and there is no guarantee that a future release won't have different assumptions, causing the upgrade to misbehave. We have seen this in practice with Quake3 and derivatives, where vendors assumed something about what may or may not be enabled during a compiled vertex array call. Most of these are just mistakes, or, occasionally, laziness.

Allowing a driver to present a non-conformant option for the user to select is an interesting question. I know that as a developer, I would get hate mail from users when a point release breaks on their whiz-bang optimized driver, just like I do with overclocked CPUs, and I would get the same "but it works with everything else!" response when I tell them to put it back to normal. On the other hand, being able to tweak around with that sort of think is fun for technically inclined users. I lean towards frowning on it, because it is a slippery slope from there down in to "cheating drivers" of the see-through- walls variety.

Quality trades are here to stay, with anti-aliasing, anisotropic texture filtering, and other options being positive trades that a user can make, and allowing various texture memory optimizations can be a very nice thing for a user trying to get some games to work well. However, it is still important that it start from a completely conformant state by default. This is one area where application naming can be used reasonably by the driver, to maintain user selected per-application modifiers.

I'm not fanatical on any of this, because the overriding purpose of software is to be useful, rather than correct, but the days of game-specific mini- drivers that can just barely cut it are past, and we should demand more from the remaining vendors.

Also, excessive optimization is the cause of quite a bit of ill user experience with computers. Byzantine code paths extract costs as long as they exist, not just as they are written.

Chapter 3

December

3.1 The Quake 2 source code is now available for download, licensed under the GPL. (Dec 21, 2001)

ftp://ftp.idsoftware.com/idstuff/source/quake2.zip

As with previous source code releases, the game data remains under the original copyright and license, and cannot be freely distributed. If you create a true total conversion, you can give (or sell) a complete package away, as long as you abide by the GPL source code license. If your projects use the original Quake 2 media, the media must come from a normal, purchased copy of the game.

I'm sure I will catch some flack about increased cheating after the source release, but there are plenty of Q2 cheats already out there, so you are already in the position of having to trust the other players to a degree. The problem is really only solvable by relying on the community to police itself, because it is a fundamentally unwinnable technical battle to make a completely cheat proof game of this type. Play with your friends.