Chris Hecker
definition six, incorporated
checker@d6.com
http://www.d6.com/users/checker

by Chris Hecker

An Open Letter to Microsoft: Do the Right Thing for the 3D Game Industry

A debate is raging in the game development community on an incredibly important topic: 3D APIs for the PC, and specifically, Direct3D versus OpenGL. This debate has its share of contentless flames, but at its core is an issue that will affect the daily lives of 3D game developers for many years to come. Being one of these developers, I care deeply about how this issue is resolved. To put it very bluntly, I believe it would be best for the 3D game industry if Microsoft canceled Direct3D Immediate Mode and put all their 3D immediate mode resources behind their OpenGL team. This article will give my rationale for that statement.

The API issue has many twists and turns. Debating it is like wrestling Jell-O; every time you think you've got it cornered, it squeezes out somewhere else. For this reason, I'm going to proceed very methodically and lay out a logical framework for my opinion. I'm sure to leave some holes through which someone can squeeze if they are intent on disagreeing with me, but I think I'll provide a vast preponderance of evidence to back up my claims. Note that most of the ideas I'll present have been stated by other people in various places, so I can't claim to have originated many of these arguments myself.

Background

First, some clarifications: When I refer to Direct3D in this article, I mean Direct3D Immediate Mode, not Retained Mode. Retained Mode will be useful to some developers, but high-end 3D games probably won't use it since they need more control over database traversal and culling. Just to be totally clear, Direct3D is a Microsoft-designed API; OpenGL was originally designed by SGI, but is now handled by an independent Architecture Review Board (the ARB, where Microsoft, SGI, and several other vendors have voting seats). In the space I have here, I'm not going to be able to describe either API, so I'm going to assume you're familiar with both (check the references at the end of the article for more information on the APIs themselves).

Next, the history: This debate has been going on for a long time. I stopped looking for the original "Direct3D versus OpenGL" Usenet posting when I found one (a reply, no less) dating all the way back to March 1996. The conventional wisdom used to be that OpenGL was inherently slow — too slow for games — and that Microsoft had to design their own API. I bought into this wisdom when I was at Microsoft (and even helped spread it there) about two or three years ago. In fact, everyone I knew was convinced OpenGL was big and slow, and the only solution for games was a new and different API. In retrospect, I realize that didn't understand the technical issues; what's worse, I didn't know that I didn't understand the issues. Even worse yet, no one within earshot understood the technical issues well enough to explain why we were all wrong. The people who really understood OpenGL were in the workstation business at this time, and didn't realize 3D games were about to become an important market segment.

Where's Physics, Part 4 ?

I'm going to take a short break from our physics series to cover a topic of great importance to the 3D game industry. I'll be back with physics next issue.

My opinion started to change last year, after a few important events. First, a bunch of people at SGI got fed up with Microsoft's game evangelists telling developers that OpenGL was inherently slow. They decided to prove that it was at least as fast as Direct3D — if not faster — with a demo at Siggraph '96. This event got everyone's attention, and indeed, focused it on Microsoft's implementation of OpenGL, which was starting to get pretty fast as well. I took an interest in the issue at this time, and started reading Usenet posts by knowledgeable 3D engineers from many different companies. Eventually, I became convinced not only that OpenGL wasn't inherently slow, but that it was inherently faster than Direct3D at the limit, for reasons I'll detail below. Next, as everyone probably knows, John Carmack of id Software released a position statement in which he made known his choice of 3D API: OpenGL. He ported Quake to OpenGL to prove that the API has what it takes for the highest end game programming. Finally, Microsoft announced plans to update Direct3D to address some of the issues people were raising.

That's the history in two paragraphs. You'll notice I keep using the term "inherent" with regards to performance. I should describe what I mean by this. When I say API A is "inherently faster" than API B, I mean that on the vast majority of hardware, the driver and application writers for A will have more opportunities for optimizations, and programs running on top of A will be faster in general, than equivalent programs running on B. So, clearly a bit of hand waving and faith is involved in saying one API is inherently faster than another. Still, I think it's possible to take knowledge of the problem domain and make a convincing case for "inherent speed" based on things like memory bandwidth and access patterns, bus speeds, existence proofs embodied in current high-end hardware, and so on. The inherent speed is important, since we don't want to run into performance ceilings imposed by our API.

This argument takes place on many levels. I'm going to show that OpenGL is inherently faster than Direct3D. However, even if they're just equal in performance, the conclusion that OpenGL is superior still holds, as you'll see. I'm reminded of one of Dave Baraff's dynamics papers that I read recently. To paraphrase, "We can always solve this problem because A is never singular, but even if it is singular we can solve the problem anyway for this other reason."

Performance

Let's start with performance. The first and most telling thing about the performance issue is that no one at Microsoft (or anywhere, actually) has been able to give me a single technical reason why Direct3D is even OpenGL's equal in 3D performance, let alone its better. I'm talking about technical engineers on the Direct3D team; when asked point blank for an architectural reason why Direct3D is (or could be) inherently faster than OpenGL, they admit they have none. The only people who routinely say Direct3D is faster for games are the Microsoft game evangelists, and they're marketing people, not technical engineers. Microsoft constantly refers to Direct3D as being "designed for games," but when pressed, no one seems to be able to come up with how that translates into actual performance improvements over OpenGL.

Although this lack of technical response is pretty damning for Direct3D, I'll still cover the specific technical reasons that OpenGL is an inherently faster immediate mode API. If you already agree that OpenGL is inherently as fast or faster than Direct3D and you just want to see all the other overwhelming reasons why OpenGL is superior, you can skip this whole next section. It's going to be heavy reading — remember, I need to try to prevent the Jell-O from squeezing out.

The performance comparison has two aspects. First, we'll compare OpenGL to Direct3D execute buffers. Then, we'll compare OpenGL to the new Direct3D DrawPrimitive API.

Execute Buffers

When comparing OpenGL's output model to Direct3D's execute buffers, we must consider the three types of 3D vertex data: static data, where the vertices of the model don't change relative to one another (like the fuselage of an airplane or the arm of a hierarchical model); dynamic data, where most vertices change every frame (like undulating water, morphing geometry, or a continuous-skinned animating figure); and finally, partially static data, which is a mix of the previous two.

STATIC DATA. For static data, OpenGL's display lists are clearly better designed than Direct3D's execute buffers, because they're opaque and noneditable. By "opaque," I mean the application doesn't know the format of the display list. By "noneditable," I mean once created, the display list's internal data cannot be changed. These two features together make it possible for driver writers to compile the display list into a format appropriate for their hardware and to upload the list onto the card, even into memory that's inaccessible to the application.

Contrast OpenGL's display lists with Direct3D's execute buffers, which have a fixed format dictated by Direct3D and are editable by the application after being created. These features make life much harder — if not impossible — for the driver writer who'd like to optimize for performance. Direct3D simply does not allow 3D hardware manufacturers the flexibility they need to optimize static vertex data rendering. In the interest of full disclosure, I should point out that Direct3D does have a little known and currently unimplemented function you can call to "optimize" the execute buffers and make them noneditable, but it's unclear whether this feature will ever be implemented. Even if it is eventually implemented, it still wouldn't allow Direct3D's theoretical performance on static data to match OpenGL's theoretical performance; OpenGL display lists have other performance-enhancing features execute buffers do not, such as the ability to invoke other display lists.

DYNAMIC DATA. 3D hardware accepts your application's rendering commands in one of two basic ways: IO or DMA. In an IO-based architecture, the 3D card memory maps its data registers and backs them up with a FIFO, allowing the data to be written directly to the card by the CPU. In DMA-based hardware, the 3D card starts up an asynchronous memory transfer to read the data directly from main memory without needing the CPU. There are, of course, hybrids that use both. Each technique has advantages and disadvantages, and which is faster is still an open topic of research (for example, the current high-end SGI hardware switches dynamically between the two, and in the PC game world, Rendition prefers DMA and 3Dfx uses IO). Since there's no consensus as to which is best, it's vitally important that the 3D API allow either or both, optimally and without preference.

Consider how OpenGL's data-formatless function-call model handles dynamically changing vertex data on both types of 3D hardware. On IO hardware, the OpenGL model allows the application to generate the vertex data and get it out to the accelerator with as little data conversion and cache pollution — and as much fine-grained parallelism — as possible. On DMA hardware, the model allows the data to be written directly to the card's DMA buffers in exactly the optimal format for the hardware, without conversion to or from an intermediate vertex format. The OpenGL model scales from rasterization-only hardware (such as current PC boards) all the way up to high-end hardware that can accept the model-space vertices directly.

Direct3D execute buffers, on the other hand, deny you maximum performance for dynamic vertex data in one of several ways. On IO-based cards, the application must write out an execute buffer full of vertices to main memory, which will then be parsed by Direct3D and written to the card. This means that your vertex data came out of your application, was reformatted into a big, bandwidth-munching main memory buffer, then read out of main memory, munged by Direct3D, and sent to the card. This extra layer of memory traffic and reformatting is death for high-throughput rendering. OpenGL sends vertex data either directly to the hardware or through a much smaller and cache-friendly single-primitive buffer.

Now consider how Direct3D execute buffers work with DMA-based designs. The best case here (or perhaps I should say the "least-worst" case) for Direct3D is if the 3D hardware can somehow directly DMA and parse the execute buffer format. Consider what is involved in this task alone; the hardware must be able to read all current and future Direct3D vertex and primitive formats, all commands, and all flags. Even the Direct3D engineers admit that this isn't likely to happen, and the hardware engineers from the PC 3D hardware companies I've talked to agree (no hardware that I know of does this, either). In addition, the application must cope with execute buffers that are different sizes on different cards, none of which might be the optimal size for your application's data.

It gets even worse for Direct3D if the DMA-based card doesn't parse the execute buffers. In this case, the application writes out an execute buffer, then the Direct3D driver must parse the buffer, write out the data again to the card's DMA buffers, and then start the DMA transfer. This is yet another layer of memory traffic over and above the IO-based card example.

Finally, some have proposed making the execute buffers writable and resident in the card's memory. In this case, the hardware designer not only needs to handle all the parsing problems mentioned previously, but also must design the hardware to allow the CPU random access to — and even reading from — the card-resident execute buffers and to have interlock protection to prevent modification of the buffers while they're being executed. No one, to my knowledge, has implemented this kind hardware.

It's clear that for dynamic vertex data, OpenGL allows much more flexibility for both the application and the 3D hardware. This flexibility directly translates into better memory access characteristics and higher performance. I should point out that a rasterization-only card will almost always deal with dynamic vertex data. This is obvious with a moment's thought; as the camera moves in 3D, the screen coordinates of all the primitives will change in each frame. Thus, this section directly applies to the vast majority of currently available commodity 3D cards for the PC.

PARTIALLY STATIC DATA. Finally, we come to partially static vertex data. If I had to pick the weakest part of my performance argument, it would be here. If you assume that you only need to update a few vertices per frame, it seems that editable execute buffers could have some benefit. However, this is only the case if the card can directly DMA and parse the buffers. If Direct3D has to parse the buffer itself, then you might as well have dynamic vertex data, and then all the previous section's criticisms apply. Also, I've yet to hear a really compelling and uncontrived example for partially static vertex data. The more the static vertex data gets, the easier it is to break up into multiple, completely static sets (OpenGL's display lists support this mixing of static data by allowing them to invoke other lists). The less static the vertex data is, the more it starts to resemble dynamic vertex data. Add to that the disadvantage that you still have the data reformatting issues with Direct3D, and I'd say it's at best a wash for both APIs on this one.

So, I think that's all she wrote for execute buffers from a performance perspective — OpenGL trounces Direct3D. I'd be very interested to hear from anyone with any other points I've missed, either for or against execute buffers.

DrawPrimitive

In some ways, it seems that Microsoft agrees with my conclusion about execute buffers. The latest feature to be added to Direct3D is the DrawPrimitive API. This is a way to draw triangles without batching them up in execute buffers. I've heard different rationalizations for adding this API, from the performance limits of execute buffers that I just discussed, to execute buffers simply being "too hard for people to use." Whatever the reason, DrawPrimitive isn't going to avoid all of Direct3D's performance problems, although it will avoid some of the memory traffic problems associated with execute buffers. Direct3D will still have the data reformatting problems, for example (the API will take Direct3D's standard vertex formats), so your application will still have to read from its database and convert its vertices into Direct3D structures, and then pass a pointer to those structures to the API. One might assume applications could use Direct3D's vertex structures internally and save a conversion, but this puts onerous constraints on the application. OpenGL doesn't have these format constraints (see the paper comparing OpenGL to PEX, referenced below, for more details on this formatting issue).

Basically, at this point, Direct3D can choose between emphasizing execute buffers and running into the performance problems I've mentioned, or it can emphasize DrawPrimitive, in which case we're stuck with an immature and poorly designed clone of OpenGL that's missing some of the architectural decisions that make OpenGL fast. Either way, game developers — and players — lose as we give away performance and get nothing in return.

Everything Else

Personally, I feel performance scalability should be the primary issue Microsoft considers when deciding which API to support for game developers; I think I've shown that OpenGL wins handily in this arena. However, there are scads of other reasons for choosing OpenGL, even if we ignore performance. I'll go into those now. The reasons are so convincing that even if OpenGL and Direct3D were somehow only evenly matched in performance, OpenGL would still be a better API for the game industry.

As I said before, no one at Microsoft was able to give me a technical reason why Direct3D was better than OpenGL. That didn't stop them from giving me a bunch of nontechnical reasons, which I'll address here.

DRIVERS. Currently (February 1997), there are more Direct3D drivers than OpenGL drivers available for Windows 95. However, this discrepancy is simply a matter of time, resources, and evangelism. All major 3D card manufacturers have now committed to doing OpenGL drivers (glQuake certainly helped motivate people on this front). Microsoft could intensify this development and shorten the gap by giving their OpenGL team more resources.

DIRECTX INTEGRATION. Direct3D is, by definition, integrated with DirectDraw. Still, nothing says OpenGL can't be integrated as well. In fact, by the time you read this, Microsoft's OpenGL-DirectDraw bindings should be announced and possibly available in beta. As with drivers, this will be nonissue in a matter of months.

EXTENSIONS. Some people at Microsoft claim they'll be able to extend Direct3D for game developers' needs faster than OpenGL could be extended. This is not only incorrect, it's backwards. OpenGL has an official and time-tested extensions mechanism, where vendors can do proprietary extensions first, then move them to multivendor extensions, and finally move them into the next OpenGL specification upon ARB approval (of which Microsoft is a voting member). Microsoft's OpenGL team has done published extensions already, as have SGI, 3Dlabs, and others. The only way for a vendor to get extensions into Direct3D is to beg Microsoft to put them in — there is no way to do it themselves. Hardware vendors always mention this as a major problem with Direct3D. What's more, OpenGL's structureless interface makes it much easier to seamlessly integrate extensions into the API; adding multitexture support to OpenGL is going to be trivial, while adding it to Direct3D is going to require creating a new vertex type.

To sum up, Microsoft can extend its OpenGL just as fast as they can extend Direct3D. As an added benefit to game developers, individual vendors can try out extensions without needing Microsoft's approval. This process is already in place and working in the OpenGL community. So, if you want to do a nifty 3Dfx-specific trick in your next game, you can do it with OpenGL, but not with Direct3D.

FOR FURTHER INFO

Usenet News
The never-ending Usenet threads are available on www.dejanews.com. Search for OpenGL and Direct3D and then hunker down for a week's worth of reading.

John Carmack's OpenGL Position Statement
http://redwood.gatsbyhouse.com/quake/jc122396.txt
If that site is down, you can find it on DejaNews. You can also find Alex St. John's reply to Carmack's statement on DejaNews.

SGI's OpenGL Page, including the 1.1 Specification
http://www.sgi.com/Technology/OpenGL

A Comparison of OpenGL and PEX
ftp://ftp.sgi.com/opengl/doc/analysis.ps.Z

Microsoft's Direct3D Immediate Mode pages
http://www.microsoft.com/msdn/sdk/platforms/doc/sdk/directx/src/directx_400.htm
http://www.microsoft.com/mediadev/graphics/drawprim.htm

Microsoft's OpenGL Docs
Microsoft has good OpenGL documentation on their web site as well, but I can't figure out how to get the direct URL. Basically, go to the following URL and select the "Documentation" tab, then "3D Graphics" in the little drop-down box, then wait for the Java applet to load. Good luck.
http://www.microsoft.com/msdn/sdk/default.htm

Brian Hook's Online 3D Papers
http://www.wksoftware.com/publications.html

OPENGL IS ONLY GOOD FOR SGI HARDWARE. I used to believe this myself, but I did a little checking. It turns out that people have done OpenGL (or its predecessor, IrisGL) on just about every type of hardware imaginable, from span renderers to full-geometry pipelines. Furthermore, consider the number of different OpenGL vendors. Check out glQuake running on a 3Dfx or an Intergraph. The more you think about it, the more you'll realize Direct3D's exposed vertex formats and execute buffers dictate the design of the hardware much more so than does OpenGL. That means there's less room for innovation by hardware designers, and again, we developers lose.

OPENGL HAS NO CAPS BITS. Direct3D has a mechanism by which you can test whether a driver supports certain features, like a Z-buffer or alpha blending (you test capabilities bits, or caps bits). In contrast, OpenGL requires that all features be implemented, whether in hardware or emulated. At first glance, it seems that Direct3D has the advantage here, but that's not the case. First, caps bits can't express the richness of 3D hardware. For example, a card that can Z-buffer and have destination alpha — but not both at the same time — is a real possibility, but you can't express that in caps bits. Second, just because a caps bit says a feature is supported doesn't mean you want to use it, since as we've all seen on this first generation of hardware, it might be slower than software (or it might not even actually be supported, given an unscrupulous vendor). The only true solution to this problem is to profile each feature at installation time and give the user a choice. You can do this equally well on either OpenGL or Direct3D.

NonDebatables

I think I've covered most of the debatable points. In addition, there are some points going for OpenGL that no one debates.

EASE OF USE AND ELEGANCE. No one argues with the fact that OpenGL is easier to use and more elegant than Direct3D. This was the central thesis of Carmack's position statement. He feels usability is even more important than the performance of the API. For more of his opinion, you should read his statement for yourself. It's in the references.

SPECIFICATION AND CONFORMANCE. It's also inarguable that OpenGL is better specified. You can read this specification on the Web; again, check the references. The spec is based on years of 3D experience, and it does a great job of balancing specificity and ambiguity, allowing hardware manufacturers room to innovate while providing software developers with a consistent, high-performance API. In addition, each OpenGL implementation must pass a battery of conformance tests. Direct3D has no comparable specification, and no are conformance tests available as of this writing.

DOCUMENTATION AND SAMPLES. OpenGL is also better documented, with many good books about it available in stores. The few Direct3D books available only cover Retained Mode in any detail. There are tons of well-written OpenGL samples available on the Web, also in contrast to Direct3D.

Summary

Can't Microsoft fix these problems? Of course they can. They can continue to change and patch Direct3D until, as Carmack says, it "sucks less." However, it'll take them years to reach the level of polish and performance that OpenGL has right now; why should we developers have to put up with it? For the same effort, Microsoft can give their OpenGL team more resources to make OpenGL even better, and the whole industry benefits and advances.

Can't we have both APIs and let them compete for mindshare? Sure, but currently the Microsoft OpenGL team is prohibited from evangelizing OpenGL to game developers because that would run counter to the current "strategy." That's not fair technical competition. Also, I showed a draft of this article to engineers from several top-echelon PC 3D hardware manufacturers to get their technical review; they all said they'd say the same thing if they weren't afraid of Microsoft and the repercussions it would cause for their companies. Again, this is not competition. And again, we developers lose. As a final reason for not having both APIs, hardware vendors are forced to spread themselves thin to support more than one API, and driver support suffers.

So, for all these reasons — for the good of the game development industry — I urge Microsoft to cancel Direct3D Immediate Mode and fully embrace OpenGL as the immediate mode game development API of choice. I also urge game developers to take a closer look at Microsoft's OpenGL. If you like what you see, use it. Remember to share your opinion with Microsoft and 3D hardware manufacturers. Finally, if you're a 3D hardware manufacturer, get your OpenGL drivers done as soon as possible.

Chris Hecker usually inserts a funny bio here at the end, but he feels strongly enough about this topic that he'll refrain for an issue. He can be reached at checker@bix.com.