John Carmack Archive - .plan (2000)

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

March 18, 2007

Contents

Chapter 1

February

1.1 Feb 23, 2000

This is a public statement that is also being sent directly to Slade at Quake-Lives regarding http://www.quakelives.com/main/ql.cgi?section=dlagreement&file=qwcl-win32/

I see both sides of this. Your goals are positive, and I understand the issues and the difficulties that your project has to work under because of the GPL. I have also seen some GPL zealots acting petty and immature towards you very early on (while it is within everyone's rights to DEMAND code under the GPL, it isn't necessarily the best attitude to take), which probably colors some of your views on the subject.

We discussed several possible legal solutions to the issues.

This isn't one of them.

While I doubt your "give up your rights" click through would hold up in court, I am positive that you are required to give the source to anyone that asks for it that got a binary from someone else. This doesn't provide the obscurity needed for a gaming level of security.

I cut you a lot of slack because I honestly thought you intended to properly follow through with the requirements of the GPL, and you were just

trying to get something fun out ASAP. It looks like I was wrong.

If you can't stand to work under the GPL, you should release the code to your last binary and give up your project. I would prefer that you continue your work, but abide by the GPL.

If necessary, I will pay whatever lawyer the Free Software Foundation reccomends to pursue this.

1.2 Feb 24, 2000

Some people took it upon themselves to remotely wreck Slade's development system. That is no more defensible than breaking into Id and smashing something.

The idea isn't to punish anyone, it is to have them comply with the license and continue to contribute. QuakeLives has quite a few happy users, and it is in everyone's best interest to have development continue. It just has to be by the rules.

Chapter 2

March

2.1 Virtualized video card local memory is The Right Thing. (Mar 07, 2000)

This is something I have been preaching for a couple years, but I finally got around to setting all the issues down in writing.

Now, the argument (and a whole bunch of tertiary information):

If you had all the texture density in the world, how much texture memory would be needed on each frame?

For directly viewed textures, mip mapping keeps the amount of referenced texels between one and one quarter of the drawn pixels. When anisotropic viewing angles and upper level clamping are taken into account, the number gets smaller. Take 1/3 as a conservative estimate.

Given a fairly aggressive six texture passes over the entire screen, that equates to needing twice as many texels as pixels. At 1024x768 resolution, well under two million texels will be referenced, no matter what the finest level of detail is. This is the worst case, assuming completely unique texturing with no repeating. More commonly, less than one million texels are actually needed.

As anyone who has tried to run certain Quake 3 levels in high quality

texture mode on an eight or sixteen meg card knows, it doesn't work out that way in practice. There is a fixable part and some more fundamental parts to the fall-over-dead-with-too-many-textures problem.

The fixable part is that almost all drivers perform pure LRU (least recently used) memory management. This works correctly as long as the total amount of textures needed for a given frame fits in the card's memory after they have been loaded. As soon as you need a tiny bit more memory than fits on the card, you fall off of a performance cliff. If you need 14 megs of textures to render a frame, and your graphics card has 12 megs available after its frame buffers, you wind up loading 14 megs of texture data over the bus every frame, instead of just the 2 megs that don't fit. Having the cpu generate 14 megs of command traffic can drop you way into the single digit frame rates on most drivers.

If an application makes reasonable effort to group rendering by texture, and there is some degree of coherence in the order of texture references between frames, much better performance can be gotten with a swapping algorithm that changes its behavior instead of going into a full thrash:

```
While ( memory allocation for new texture fails )
    Find the least recently used texture.
    If the LRU texture was not needed in the previous frame,
        Free it
    Else
        Free the most recently used texture that isn't bound to an active texture unit
```

Freeing the MRU texture seems counterintuitive, but what it does is cause the driver to use the last bit of memory as a sort of scratchpad that gets constantly overwritten when there isn't enough space. Pure LRU plows over all the other textures that are very likely going to be needed at the beginning of the next frame, which will then plow over all the textures that were loaded on top of them.

If an application uses textures in a completely random order, any given replacement policy has the some effect..

Texture priority for swapping is a non-feature. There is NO benefit to attempting to statically prioritize textures for swapping. Either a texture

is going to be referenced in the next frame, or it isn't. There aren't any useful gradations in between. The only hint that would be useful would be a notice that a given texture is not going to be in the next frame, and that just doesn't come up very often or cover very many texels.

With the MRU-on-thrash texture swapping policy, things degrade gracefully as the total amount of textures increase but due to several issues, the total amount of textures calculated and swapped is far larger than the actual amount of texels referenced to draw pixels.

The primary problem is that textures are loaded as a complete unit, from the smallest mip map level all the way up to potentially a 2048 by 2048 top level image. Even if you are only seeing 16 pixels of it off in the distance, the entire 12 meg stack might need to be loaded.

Packing can also cause some amount of wasted texture memory. When you want to load a two meg texture, it is likely going to require a lot more than just two megs of free texture memory, because a lot of it is going to be scattered around in 8k to 64k blocks. At the pathological limit, this can waste half your texture memory, but more reasonably it is only going to be 10% or so, and cause a few extra texture swap outs.

On a frame at a time basis, there are often significant amounts of texels even in referenced mip levels that are not seen. The back sides of characters, and large textures on floors can often have less than 50% of their texels used during a frame. This is only an issue as they are being swapped in, because they will very likely be needed within the next few frames. The result is one big hitch instead of a steady loading.

There are schemes that can help with these problems, but they have costs.

Packing losses can be addressed with compaction, but that has rarely proven to be worthwhile in the history of memory management. A 128-bit graphics accelerator could compact and sort 10 megs of texture memory in about 10 msec if desired.

The problems with large textures can be solved by just not using large textures. Both packing losses, and non- referenced texels can be reduced by chopping everything up into 64x64 or 128x128 textures. This requires preprocessing, adds geometry, and requires messy overlap of the textures

to avoid seaming problems.

It is possible to estimate which mip levels will actually be needed and only swap those in. An application can't calculate exactly the mip map levels that will be referenced by the hardware, because there are slight variations between chips and the slope calculation would add significant processing overhead. A conservative upper bound can be taken by looking at the minimum normal distance of any vertex referencing a given texture in a frame. This will overestimate the required textures by 2x or so and still leave a big hit when the top mip level loads for big textures, but it can allow giant cathedral style scenes to render without swapping.

Clever programmers can always work harder to overcome obstacles, but in this case, there is a clear hardware solution that gives better performance than anything possible with software and just makes everyone's lives easier: virtualize the card's view of its local memory.

With page tables, address fragmentation isn't an issue, and with the graphics rasterizer only causing a page load when something from that exact 4k block is needed, the mip level problems and hidden texture problems just go away. Nothing sneaky has to be done by the application or driver, you just manage page indexes.

The hardware requirements are not very heavy. You need translation lookaside buffers (TLB) on the graphics chip, the ability to automatically load the TLB from a page table set up in local memory, and the ability to move a page from AGP or PCI into graphics memory and update the page tables and reference counts. You don't even need that many TLB, because graphics access patterns don't hop all over the place like CPU access can. Even with only a single TLB for each texture bilerp unit, reloads would only account for about 1/32 of the memory access if the textures were 4k blocked. All you would really want at the upper limit would be enough TLB for each texture unit to cover the texels referenced on a typical rasterization scan line.

Some programmers will say "I don't want the system to manage the textures, I want full control!" There are a couple responses to that. First, a page level management scheme has flexibility that you just can't get with a software only scheme, so it is a set of brand new capabilities. Second, you can still just choose to treat it as a fixed size texture buffer and

manage everything yourself with updates. Third, even if it WAS slower than the craftiest possible software scheme (and I seriously doubt it), so much of development is about willingly trading theoretical efficiency for quicker, more robust development. We don't code overlays in assembly language any more..

Some hardware designers will say something along the lines of "But the graphics engine goes idle when you are pulling the page over from AGP!" Sure, you are always better off to just have enough texture memory and never swap, and this feature wouldn't let you claim any more megapixels or megatris, but every card winds up not having enough memory at some point. Ignoring those real world cases isn't helping your customers. In any case, it goes idle a hell of a lot less than if you were loading the entire texture over the command fifo.

3Dlabs is supposed to have some form of virtual memory management in the permedia 3, but I am not familiar with the details (if anyone from 3dlabs wants to send me the latest register specs, I would appreciate it!).

A mouse controlled first person shooter is fairly unique in how quickly it can change the texture composition of a scene. A 180-degree snap turn can conceivably bring in a completely different set of textures on a subsequent frame. Almost all other graphics applications bring textures in at a much steadier pace.

So, given that 180-degree snap turn to a completely different and uniquely textured scene, what would be the worst case performance? An AGP 2x bus is theoretically supposed to have over 500 mb/sec of bandwidth. It doesn't get that high in practice, but linear 4k block reads would give it the best possible conditions, and even at 300 mb/sec, reloading the entire texture working set would only take 10 msec.

Rendering is not likely to be buffered sufficiently to overlap appreciably with page loading, and the command transport for a complex scene will take significant time by itself, so it shows that a worst case scene will often not be able to be rendered in 1/60th of a second.

This is roughly the same lower bound that you get from a chip texturing directly from AGP memory. A direct AGP texture gains the benefit of fine-grained rendering overlap, but loses the benefit of subsequent references

being in faster memory (outside of small on-chip caches). A direct AGP texture engine doesn't have the higher upper bounds of a cached texture engine, though. It's best and worst case are similar (generally a good thing), but the cached system can bring several times more bandwidth to bear when it isn't forced to swap anything in.

The important point is that the lower performance bound is almost an order of magnitude faster than swapping in the textures as a unit by the driver.

If you just positively couldn't deal with the chance of that much worst case delay, some form of mip level biasing could be made to kick in, or you could try and do pre-touching, but I don't think it would ever be worth it. The worst imaginable case is acceptable, and you just won't hit that case very often.

Unless a truly large number of TLB are provided, the textures would need to be blocked. The reason is that with a linear texture, a 4k page maps to only a couple scan lines on very large textures. If you are going with the grain you get great reuse, but if you go across it, you wind up referencing a new page every couple texel accesses. What is wanted is an addressing mechanism that converts a 4k page into a square area in the texture, so the page access is roughly constant for all orientations. There is also a benefit from having a 128 bit access also map to a square block of pixels, which several existing cards already do. The same interleaving-of-low-order-bits approach can just be extended a few more bits.

Dealing with blocked texture patterns is a hassle for a driver writer, but most graphics chips have a host blit capability that should let the chip deal with changing a linear blit into blocked writes. Application developers should never know about it, in any case.

There are some other interesting things that could be done if the page tables could trigger a cpu interrupt in addition to being automatically backed by AGP or PCI memory. Textures could be paged in directly from disk for truly huge settings, or decompressed from jpeg blocks, or even procedurally generated. Even the size limits of the AGP aperture could usefully be avoided if the driver wanted to manage each page's allocation.

Aside from all the basic swapping issue, there are a couple of other hard-

ware trends that push things this way.

Embedded dram should be a driving force. It is possible to put several megs of extremely high bandwidth dram on a chip or die with a video controller, but won't be possible (for a while) to cram a 64 meg geforce in. With virtualized texturing, the major pressure on memory is drastically reduced. Even an 8mb card would be sufficient for 16 bit 1024x768 or 32 bit 800x600 gaming, no matter what the texture load.

The only thing that prevents a geometry processor based card from turning almost any set of commands in a display list into a single static dma buffer is the fact that textures may be swapped in and out, causing the register programming in the buffer to be wrong. With virtual texture addressing, a texture's address never changes, and an arbitrarily complex model can be described in a static dma buffer.

2.2 Seumas McNally (Mar 27, 2000)

Two years ago, Id was contacted by the Startlight Foundation, an organization that tries to grant wishes to seriously ill kids. (http://www.starlight.org)

There was a young man with Hodgkin's Lymphoma that, instead of wanting to go to Disneyland or other traditional wishes, wanted to visit Id and talk with me about programming.

It turned out that Seumas McNally was already an accomplished developer. His family company, Longbow Digital Arts (http://www.longbowdigitalarts.com), had been doing quite respectably selling small games directly over the internet. It bore a strong resemblance to the early shareware days of Apogee and Id.

We spent the evening talking about graphics programmer things - the relative merits of voxels and triangles, procedurally generated media, level of detail management, API and platforms.

We talked at length about the balance between technology and design, and all the pitfalls that lie in the way of shipping a modern product.

We also took a dash out in my ferrari, thinking "this is going to be the best excuse a cop will ever hear if we get pulled over".

Longbow continued to be successful, and eventually the entire family was working full time on "Treadmarks", their new 3D tank game.

Over email about finishing the technology in Treadmarks, Seumas once said "I hope I can make it". Not "be a huge success" or "beat the competition". Just "make it".

That is a yardstick to measure oneself by.

It is all too easy to lose your focus or give up with just the ordinary distractions and disappointments that life brings. This wasn't ordinary. Seumas had cancer. Whatever problems you may be dealing with in your life, they pale before having problems drawing your next breath.

He made it.

Treadmarks started shipping a couple months ago, and was entered in the Independent Games Festival at the Game Developer's Conference this last month. It came away with the awards for technical excellence, game design, and the grand prize.

I went out to dinner with the McNally family the next day, and had the opportunity to introduce Anna to them. One of the projects at Anna's new company, Fountainhead Entertainment (http://www.fountainheadent.com), is a documentary covering gaming, and she had been looking forward to meeting Seumas after hearing me tell his story a few times. The McNallys invited her to bring a film crew up to Canada and talk with everyone whenever she could.

Seumas died the next week.

I am proud to have been considered an influence in Seumas' work, and I think his story should be a good example for others. Through talent and determination, he took something he loved and made a success out of it in many dimensions.

http://www.gamedev.net/community/memorial/seumas/ for more information.

Chapter 3

April

3.1 Apr 06, 2000

Whenever I start a new graphics engine, I always spend a fair amount of time flipping back through older graphics books. It is always interesting to see how your changed perspective with new experience impacts your appreciation of a given article.

I was skimming through Jim Blinn's "A Trip Down The Graphics Pipeline" tonight, and I wound up laughing out loud twice.

From the book:

P73: I then empirically found that I had to scale by -1 in x instead of in z, and also to scale the xa and xf values by -1. (Basically I just put in enough minus signs after the fact to make it work.) Al Barr refers to this technique as "making sure you have made an even number of sign errors."

P131: The only lines that generate w=0 after clipping are those that pass through the z axis, the valley of the trough. These lines are lines that pass exactly through the eyepoint. After which you are dead and don't care about divide-by-zero errors.

If you laughed, you are a graphics geek.

My first recollection of a Jim Blinn article many years ago was my skimming over it and thinking "My god, what ridiculously picky minutia." Over the last couple years, I found myself haranguing people over some fairly picky issues, like the LSB errors with cpu vs rasterizer face culling and screen edge clipping with guard band bit tests. After one of those pitches, I quite distinctly thought to myself "My god, I'm turning into Jim Blinn!" :)

3.2 We need more bits per color component in our 3D accelerators. (Apr 29, 2000)

I have been pushing for a couple more bits of range for several years now, but I now extend that to wanting full 16 bit floating point colors throughout the graphics pipeline. A sign bit, ten bits of mantissa, and five bits of exponent (possibly trading a bit or two between the mantissa and exponent). Even that isn't all you could want, but it is the rational step.

It is turning out that I need a destination alpha channel for a lot of the new rendering algorithms, so intermediate solutions like 10/12/10 RGB formats aren't a good idea. Higher internal precision with dithering to 32 bit pixels would have some benefit, but dithered intermediate results can easily start piling up the errors when passed over many times, as we have seen with 5/6/5 rendering.

Eight bits of precision isn't enough even for full range static image display. Images with a wide range usually come out fine, but restricted range images can easily show banding on a 24-bit display. Digital television specifies 10 bits of precision, and many printing operations are performed with 12 bits of precision.

The situation becomes much worse when you consider the losses after multiple operations. As a trivial case, consider having multiple lights on a wall, with their contribution to a pixel determined by a texture lookup. A single light will fall off towards 0 some distance away, and if it covers a large area, it will have visible bands as the light adds one unit, two units, etc. Each additional light from the same relative distance stacks its con-

tribution on top of the earlier ones, which magnifies the amount of the step between bands: instead of going 0,1,2, it goes 0,2,4, etc. Pile a few lights up like this and look towards the dimmer area of the falloff, and you can believe you are back in 256-color land.

There are other more subtle issues, like the loss of potential result values from repeated squarings of input values, and clamping issues when you sum up multiple incident lights before modulating down by a material.

Range is even more clear cut. There are some values that have intrinsic ranges of 0.0 to 1.0, like factors of reflection and filtering. Normalized vectors have a range of -1.0 to 1.0. However, the most central quantity in rendering, light, is completely unbounded. We want a LOT more than a 0.0 to 1.0 range. Q3 hacks the gamma tables to sacrifice a bit of precision to get a 0.0 to 2.0 range, but I wanted more than that for even primitive rendering techniques. To accurately model the full human sensable range of light values, you would need more than even a five bit exponent.

This wasn't much of an issue even a year ago, when we were happy to just cover the screen a couple times at a high framerate, but realtime graphics is moving away from just "putting up wallpaper" to calculating complex illumination equations at each pixel. It is not at all unreasonable to consider having twenty textures contribute to the final value of a pixel. Range and precision matter.

A few common responses to this pitch:

"64 bits per pixel??? Are you crazy???" Remember, it is exactly the same relative step as we made from 16 bit to 32 bit, which didn't take all that long.

Yes, it will be slower. That's ok. This is an important point: we can't continue to usefully use vastly greater fill rate without an increase in precision. You can always crank the resolution and multisample anti-alaising up higher, but that starts to have diminishing returns well before you use of the couple gigatexels of fill rate we are expected to have next year. The cool and interesting things to do with all that fill rate involves many passes composited into less pixels, making precision important.

"Can we just put it in the texture combiners and leave the framebuffer

at 32 bits?" No. There are always going to be shade trees that overflow a given number of texture units, and they are going to be the ones that need the extra precision. Scales and biases between the framebuffer and the higher precision internal calculations can get you some mileage (assuming you can bring the blend color into your combiners, which current cards can't), but its still not what you want. There are also passes which fundamentally aren't part of a single surface, but still combine to the same pixels, as with all forms of translucency, and many atmospheric effects.

"Do we need it in textures as well?" Not for most image textures, but it still needs to be supported for textures that are used as function look up tables.

"Do we need it in the front buffer?" Probably not. Going to a 64 bit front buffer would probably play hell with all sorts of other parts of the system. It is probably reasonable to stay with 32 bit front buffers with a blit from the 64 bit back buffer performing a lookup or scale and bias operation before dithering down to 32 bit. Dynamic light adaptation can also be done during this copy. Dithering can work quite well as long as you are only performing a single pass.

I used to be pitching this in an abstract "you probably should be doing this" form, but two significant things have happened that have moved this up my hit list to something that I am fairly positive about.

Mark Peercy of SGI has shown, quite surprisingly, that all Renderman surface shaders can be decomposed into multi-pass graphics operations if two extensions are provided over basic OpenGL: the existing pixel texture extension, which allows dependent texture lookups (matrox already supports a form of this, and most vendors will over the next year), and signed, floating point colors through the graphics pipeline. It also makes heavy use of the existing, but rarely optimized, copyTexSubImage2D functionality for temporaries.

This is a truly striking result. In retrospect, it seems obvious that with adds, multiplies, table lookups, and stencil tests that you can perform any computation, but most people were working under the assumption that there were fundamentally different limitations for "realtime" renderers vs offline renderers. It may take hundreds or thousands of passes, but

it clearly defines an approach with no fundamental limits. This is very important. I am looking forward to his Siggraph paper this year.

Once I set down and started writing new renderers targeted at GeForce level performance, the precision issue has started to bite me personally. There are quite a few times where I have gotten visible banding after a set of passes, or have had to worry about ordering operations to avoid clamping. There is nothing like actually dealing with problems that were mostly theoretical before..

64 bit pixels. It is The Right Thing to do. Hardware vendors: don't you be the company that is the last to make the transition.

3.2. WE NEED MORE BITS PER COLOR COMPONENT IN OUR 3D ACCELERATORS. (APR 29, 2000)

Chapter 4

May

4.1 May 08, 2000

The .qc files for quake1/quakeworld are now available under the GPL in source/qw-qc.tar.gx on out ftp site. This was an oversight on my part in the original release.

Thanks to the QuakeForge team for doing the grunt work of the preparation.

4.2 May 09, 2000

And the Q1 utilities are now also available under the GPL in source/q1tools_gpl.tgz.

4.3 May 14, 2000

I stayed a couple days after E3 to attend the SORAC amateur rocket launch. I have provided some sponsorship to two of the teams competing for the CATS (Cheap Access to Space) rocketry prize, and it was a nice opportu-

nity to get out and meet some of the people.

It is interesting how similar the activity is around an experimental rocket launch, going to a race track with an experimental car, and putting out a beta version of new software is. Lots of "twenty more minutes!", and lots of well-wishers waiting around while the people on the critical path sweat over what they are doing.

Mere minutes before we absolutely, positively needed to leave to catch our plane flight, they started the countdown. The rocket launched impressively, but broke apart at a relatively low altitude. Ouch. It was a hybrid, so there wasn't really an explosion, but watching the debris rain down wasn't very heartening. Times like that, I definitely appreciate working in software. "Run it again, with a breakpoint!"

Note to self: pasty-skinned programmers ought not stand out in the Mojave desert for multiple hours.

http://www.space-frontier.org/Events/CATSPRIZE_1/
http://www.energyrs.com/sorac/sorac.htm
http://www.jpaerospace.com/

4.4 May 17, 2000

I have gotten a lot of requests for comments on the latest crop of video cards, so here is my initial technical evaluation. We have played with some early versions, but this is a paper evaluation. I am not in a position to judge 2D GDI issues or TV/DVD issues, so this is just 3D commentary.

Nvidia Marketing silliness: saying "seven operations on a pixel" for a dual texture chip. Yes, I like NV_register_combiners a lot, but come on..

The DDR GeForce is the reigning champ of 3D cards. Of the shipping boards, it is basically better than everyone at every aspect of 3D graphics, and pioneered some features that are going to be very important: signed pixel math, dot product blending, and cubic environment maps.

The GeForce2 is just a speed bumped GeForce with a few tweaks, but

that's not a bad thing. Nvidia will have far and away the tightest drivers for quite some time, and that often means more than a lot of new features in the real world.

The nvidia register combiners are highly programmable, and can often save a rendering pass or allow a somewhat higher quality calculation, but on the whole, I would take ATI's third texture for flexibility.

Nvidia will probably continue to hit the best framerates in benchmarks at low resolution, because they have flexible hardware with geometry acceleration and well-tuned drivers.

GeForce is my baseline for current rendering work, so I can wholeheartedly recommend it.

ATI Marketing silliness: "charisma engine" and "pixel tapestry" are silly names for vertex and pixel processing that are straightforward improvements over existing methods. Sony is probably to blame for starting that.

The Radeon has the best feature set available, with several advantages over GeForce:

A third texture unit per pixel
Three dimensional textures
Dependent texture reads (bump env map)
Greater internal color precision.
User clip planes orthogonal to all rasterization modes.
More powerful vertex blending operations.
The shadow id map support may be useful, but my work with shadow buffers have shown them to have significant limitations for global use in a game.

On paper, it is better than GeForce in almost every way except that it is limited to a maximum of two pixels per clock while GeForce can do four. This comes into play when the pixels don't do as much memory access, for example when just drawing shadow planes to the depth/stencil buffer, or when drawing in roughly front to back order and many of the later pixels depth fail, avoiding the color buffer writes.

Depending on the application and algorithm, this can be anywhere from basically no benefit when doing 32 bit blended multi-pass, dual texture

rendering to nearly double the performance for 16 bit rendering with compressed textures. In any case, a similarly clocked GeForce(2) should somewhat outperform a Radeon on today's games when fill rate limited. Future games that do a significant number of rendering passes on the entire world may go back in ATI's favor if they can use the third texture unit, but I doubt it will be all that common.

The real issue is how quickly ATI can deliver fully clocked production boards, bring up stable drivers, and wring all the performance out of the hardware. This is a very different beast than the Rage128. I would definitely recommend waiting on some consumer reviews to check for teething problems before upgrading to a Radeon, but if things go well, ATI may give nvidia a serious run for their money this year.

3DFX Marketing silliness: Implying that a voodoo 5 is of a different class than a voodoo 4 isn't right. Voodoo 4 max / ultra / SLI / dual / quad or something would have been more forthright.

Rasterization feature wise, voodoo4 is just catching up to the original TNT. We finally have 32 bit color and stencil. Yeah.

There aren't any geometry features.

The T buffer is really nothing more than an accumulation buffer that is averaged together during video scanout. This same combining of separate buffers can be done by any modern graphics card if they are set up for it (although they will lose two bits of color precision in the process). At around 60 fps there is a slight performance win by doing it at video scannout time, but at 30 fps it is actually less memory traffic to do it explicitly. Video scan tricks also usually don't work in windowed modes.

The real unique feature of the voodoo5 is subpixel jittering during rasterization, which can't reasonably be emulated by other hardware. This does indeed improve the quality of anti-aliasing, although I think 3dfx might be pushing it a bit by saying their 4 sample jittering is as good as 16 sample unjittered.

The saving grace of the voodoo5 is the scalability. Because it only uses SDR ram, a dual chip Voodoo5 isn't all that much faster than some other single chip cards, but the quad chip card has over twice the pixel fill

rate of the nearest competitor. That is a huge increment. Voodoo5 6000 should win every benchmark that becomes fill rate limited.

I haven't been able to honestly recommend a voodoo3 to people for a long time, unless they had a favorite glide game or wanted early linux Xfree 4.0 3D support. Now (well, soon), a Voodoo5 6000 should make all of today's games look better than any other card. You can get over twice as many pixel samples, and have them jittered and blended together for anti-aliasing.

It won't be able to hit Q3 frame rates as high as GeForce, but if you have a high end processor there really may not be all that much difference for you between 100fps and 80fps unless you are playing hardcore competitive and can't stand the occasional drop below 60fps.

There are two drawbacks: it's expensive, and it won't take advantage of the new rasterization features coming in future games. It probably wouldn't be wise to buy a voodoo5 if you plan on keeping it for two years.

Chapter 5

June

5.1 Well, this is going to be an interesting .plan update. (Jun 01, 2000)

Most of this is not really public business, but if some things aren't stated explicitly, it will reflect unfairly on someone.

As many people have heard discussed, there was quite a desire to remake DOOM as our next project after Q3. Discussing it brought an almost palpable thrill to most of the employees, but Adrian had a strong enough dislike for the idea that it was shot down over and over again.

Design work on an alternate game has been going on in parallel with the mission pack development and my research work.

Several factors, including a general lack of enthusiasm for the proposed plan, the warmth that Wolfenstien was met with at E3, and excitement about what we can do with the latest rendering technology were making it seem more and more like we weren't going down the right path.

I discussed it with some of the other guys, and we decided that it was important enough to drag the company through an unpleasant fight over it.

An ultimatum was issued to Kevin and Adrian(who control $>$ 50% of the

company): We are working on DOOM for the next project unless you fire us.

Obviously no fun for anyone involved, but the project direction was changed, new hires have been expedited, and the design work has begun.

It wasn't planned to announce this soon, but here it is: We are working on a new DOOM game, focusing on the single player game experience, and using brand new technology in almost every aspect of it. That is all we are prepared to say about the game for quite some time, so don't push for interviews. We will talk about it when things are actually built, to avoid giving misleading comments.

It went smoother than expected, but the other shoe dropped yesterday.

Kevin and Adrian fired Paul Steed in retaliation, over my opposition.

Paul has certainly done things in the past that could be grounds for dismissal, but this was retaliatory for him being among the "conspirators".

I happen to think Paul was damn good at his job, and that he was going to be one of the most valuable contributors to DOOM.

We need to hire two new modeler/animator/cinematic director types. If you have a significant commercial track record in all three areas, and consider yourself at the top of your field, send your resume to Kevin Cloud.