John Carmack Archive - .plan (1996)

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

March 18, 2007

Contents

Chapter 1

July

1.1 Quake is out. Finally. (Jul 01, 1996)

I think the end product is damn good, and I am proud of my 18+ months of sweating over it. I hope lots of you enjoy the shareware levels enough to register :-)

While there will probably not be any about-faces in technical decisions, Quake will definately be growing and expanding over the next year or so as we gauge the user community reaction and release updates and the inevitable sequal.

Some work that will be going on in the near future:

Fixing any serious problems with Quake 0.91.
Release of tools and source code for user prog coding and level building.
Writing a new map editor for win/NT + open GL Porting Quake to native win32 + DirectX
Porting Quake on the metal to some 3D accelerator boards.
Writing a proper full radiosity replacement for the light utility.
Reasonable user suggested improvements (keep it in the discussion groups please, I get enough mail as it is)
Speed enhancements in the polygon model path.
Adding support for various gadgets – HMDs, controllers, maybe other

sound cards, etc.
And of course... The Next Generation Technology. (no, I won't give you a code name to talk about!)

Vacation? I don't need a vacation!

John Carmack Id Software

Chapter 2

August

2.1 Here is The New Plan (Aug 02, 1996)

I copied off the quake codebase and set about doing some major improvements. The old v1.01 codebase will still be updated to fix bugs with the current version, but I didn't want to hold back from fixing things properly even if it involves some major changes.

I am focusing on the internet play aspect of the game. While I can lay out a grand clean-sheet-of-paper design, I have chosen to pursue something of a limited enough scope that I can expect to start testing it around the end of the month (august). I still have my grand plans for the future, but I want to get some stuff going NOW.

QuakeWorld.

The code I am developing right now is EXCLUSIVELY for internet play. It will be rolled back into the single player game sometime along the road to Quake 2 (or whatever it turns out to be called), but the experimental QuakeWorld release will consist of seperate programs for the client and the server. They will use the same data as the current registered quake, so the only thing that will be distributed is new executables (they will peacefully coexist with current quake).

There will be a single master server running here at id. Whenever anyone

starts up a server, it will register itself with the master server, and whenever a client wants to start a game, it will inquire with the master to find out which servers are available.

Users will have a persistant account, and all frags on the entire internet will be logged. I want us to be able to give a global ranking order of everyone playing the game. You should be able to say, "I am one of the ten best QuakeWorld players in existance", and have the record to back it up. There are all sorts of other cool stats that we could mine out of the data: greatest frags/minute, longest uninterrupted quake game, cruelest to newbies, etc, etc.

For the time being, this is just my pet research project. The new exes will only work with registered Quake, so I can justify it as a registration incentive (don't pirate!).

If it looks feasable, I would like to see internet focused gaming become a justifiable biz direction for us. Its definately cool, but it is uncertain if people can actually make money at it. My halfway thought out proposal for a biz plan is that we let anyone play the game as an anonymous newbie to see if they like it, but to get their name registered and get on the ranking list, they need to pay $10 or so. Newbies would be automatically kicked from servers if a paying customer wants to get on. Sound reasonable?

Technical improvements.

The game physics is being reworked to make it faster and more uniform. Currently, a p90 dedicated server is about 50% loaded with eight players. The new network code causes a higher cpu load, so I am trying to at least overbalance that, and maybe make a little headway. A single p6-200 system should be able to run around ten simultanious eight player servers. Multiple servers running on a single machine will work a lot better with the master server automatically dealing with different port adresses behind the client's back.

A couple subtle features are actually going away. The automatic view tilting on slopes and stairs is buggy in v1.01, and over a couple hundred millisecond latancy connection, it doesn't usually start tilting until you are allready on a different surface, so I just ripped it out entirely. A few other

non-crucial game behaviors are also being cut in the interest of making the physics easier to match on the client side.

I'm going to do a good chat mode.

Servers will have good access control lists. If somebody manages to piss off the entire community, we could even ban them at the master server.

The big difference is in the net code. While I can remember and justify all of my decisions about networking from DOOM through Quake, the bottom line is that I was working with the wrong basic assumptions for doing a good internet game. My original design was targeted at $<$ 200ms connection latencies. People that have a digital connection to the internet through a good provider get a pretty good game experience. Unfortunately, 99% of the world gets on with a slip or ppp connection over a modem, often through a crappy overcrowded ISP. This gives 300+ ms latencies, minimum. Client. User's modem. ISP's modem. Server. ISP's modem. User's modem. Client. God, that sucks.

Ok, I made a bad call. I have a T1 to my house, so I just wasn't familliar with PPP life. I'm adressing it now.

The first move was to scrap the current net code. It was based on a reliable stream as its original primitive (way back in qtest), then was retrofited to have an unreliable sideband to make internet play feasable. It was a big mess, so I took it out and shot it. The new code has the unreliable packet as its basic primitive, and all the complexities that entails is now visible to the main code instead of hidden under the net api. This is A Good Thing. Goodbye phantom unconnected players, messages not getting through, etc.

The next move was a straightforward attack on latency. The communications channel is not the only thing that contributes to a latent response, and there was some good ground to improve on.

In a perfect environment, the instant you provided any input (pressed a key, moved a mouse, etc) you would have feedback on the screen (or speaker) from the action.

In the real world, even single player games have latency.

A typical game loop goes something like: Read user input. Simulate the world. Render a new graphics scene. Repeat.

If the game is running 15 frames a second, that is 66 ms each frame. The user input will arive at a random point in the frame, so it will be an average of 33 ms before the input is even looked at. The input is then read, and 66 more ms pass before the result is actually displayed to the user, for a total of nearly 100 ms of latency, right on your desktop. (you can even count another 8 ms or so for raster refresh if you want to get picky).

The best way to adress that latency is to just make the game run faster if possible. If the screen was sized down so that the game ran 25 fps, the latency would be down to 60ms. There are a few other things that can be done to shave a bit more off, like short circuiting some late braeking information (like view angles) directly into the refresh stage, bypassing the simulation stage.

The bearing that this all has on net play, aside from setting an upper limit on performance, is that the current Quake servers have a similar frame cycle. They had to, to provide -listen server support. Even when you run a dedicated server, the model is still: fetch all input, process the world, send updates out to all clients. The default server framerate is 20 fps (50 ms). You can change this by adjusting the sys_ticrate cvar, but there are problems either way. If you ask for more fps from the server, you may get less latency, but you would almost certainly overcommit the bandwidth of a dialup link, resulting in all sorts of unwanted buffering in the routers and huge multi thousand ms latency times as things unclog (if they ever do).

The proper way to address this is by changing the server model from a game style loop to a fileserver/database style loop.

Instead of expecting everyone's messages to be dealt with at once, I now deal with each packet as it comes in. That player alone is moved forward in time, and a custom response is sent out in very short order. The rest of the objects in the world are spread out between the incoming packets. There are a lot of issues that that brings up. Time is no longer advancing uniformly for all objects in the world, which can cause a lot of problems.

It works, though! The average time from a packet ariving at the system

to the time a response is sent back is down to under 4ms, as opposed to over 50 with the old dedicated servers.

Another side benefit is that the server never blindly sends packets out into the void, they must be specifically asked for (note that this is NOT a strict request/reply, because the client is streaming request without waiting for the replies).

I am going to be adding bandwidth estimation to help out modem links. If quake knows that a link is clogged up, it can choose not to send anything else, which is far, far better than letting the network buffer everything up or randomly drop packets. A dialup line can just say "never send more than 2000 bytes a second in datagrams", and while the update rate may drop in an overcommited situation, the latency will never pile up like it can with the current version of quake.

The biggest difference is the addition of client side movement simulation.

I am now allowing the client to guess at the results of the users movement until the authoritative response from the server comes through. This is a biiiig architectural change. The client now needs to know about solidity of objects, friction, gravity, etc. I am sad to see the elegent client-as-terminal setup go away, but I am practical above idealistic.

The server is still the final word, so the client is allways repredicting it's movement based off of the last known good message from the server.

There are still a lot of things I need to work out, but the basic results are as hoped for: even playing over a 200+ ms latency link, the player movement feels exactly like you are playing a single player game (under the right circumstances – you can also get it to act rather weird at the moment).

The latency isn't gone, though. The client doesn't simulate other objects in the world, so you apear to run a lot closer to doors before they open, and most noticably, projectiles from your weapons seem to come out from where you were, instead of where you are, if you are strafing sideways while you shoot.

An interesting issue to watch when this gets out is that you won't be able

to tell how long the latency to the server is based on your movement, but you will need to lead your opponents differently when shooting at them.

In a clean sheet of paper redesign, I would try to correct more of the discrepencies, but I think I am going to have a big enough improvement coming out of my current work to make a lot of people very happy.

2.2 Romero is now gone from id. (Aug 08, 1996)

There will be no more grandiose statements about our future projects.

I can tell you what I am thinking, and what I am trying to acomplish, but all I promise is my best effort.

2.3 QuakeWorld structural addendum (Aug 10, 1996)

After hearing many arguments against the single master server, ranging from coherent and well reasoned to paranoid whining, I now agree that the single global master server isn't sufficient.

During the R&D phase, there will still be only the single server, but after all the kinks get worked out, I will allow a small number of sites to run private master servers. This will not be a general release, but only to properly licensed third parties. That will still allow me to collect my 100% coverage data, and it will prevent a single network/computer/software failure from preventing all QuakeWorld play.

QuakeWorld technical addendum:

I am reining in the client side prediction to a fairly minimal amount. It has too many negative effects in different circumstances. The problem of running away from or in front of your missiles was so bad that I considered simulating the missiles on the client side, but that is the second step on a slippery slope. If just the missiles were simulated, then a missile

would fire through an enemy until the server informed you it exploded on them. Then you consider simulating interactions, but then you have to guess at other player inputs (which is hopeless)...

Lesson learned: Simulating 300 ms on the client side in a Quake style game is just out of the question. It probably works fine for flight sim or driving sims, but not in out twitch reaction games.

I am currently using client side simulation to smooth out the beat frequency interactions between server packet arrival and client frame times. In the shipping version of Quake, some latency was introduced on purpose to keep the displayed frame simulation time between the last two packets from the server so that the variability in arrival time could be smoothed out. In QuakeWorld, I am always starting with the most current packet, and using simulation to smooth out the variability. This $<$ 100ms of client side motion seems to be very manageable, and cuts out some real latency as well as doing the gueswork.

It looks like I am going to split the QuakeWorld client into multiple threads to reduce the avg 1/2 frame latency between input and packet sending. This is also a step towards building a multi-threaded Quake renderer, which will let multi-cpu NT machines render twice as fast. Lets hope the windows thread scheduler is decent...

I have been cutting down the message sizes a bit here and there as well. On a modem link, every couple bytes I save translates into a millisecond of latency saved. I cut an average of 17 bytes from the server to client and 8 from the client to server today.

2.4 Aug 12, 1996

Apogee's Prey team (and Duke's Levelord) leave 3drealms to work with Quake technology as Hipnotic Interactive.

:-)

2.5 Aug 13, 1996

I am considering increasing the default sv_friction value for QuakeWorld from 4 to 6 or 8.

It might take a little getting used to, but I think it gives more precise control for wide area network play.

If anyone wants to run some experiments with different friction levels on a current Quake server, I would be interested in hearing some feedback.

2.6 The remote server console commands are fully implemented for QuakeWorld. (Aug 17, 1996)

To allow remote administration, the server must set the "password" cvar. By default, remote administration is turned off.

On a client, if you set the "password" cvar to the same value, you can issue "rcon" commands to the remote server :

rcon < server command> < arguments> ...

You can go to different levels, shut the server down, change cvars, ban people, etc. The output from the command is redirected over the net and will be echoed on the remote console.

You can also execute commands without even connecting to the server (if it was full) by setting the "rconadr" cvar to the full internet address (including port) of the system you want to administer.

2:00 in the morning:

My testarossa snapped another input shaft (the third time). damn dman damn. > 1000 HP is bad for your drivetrain.

2.7 PACKET FILTERING (Aug 18, 1996)

QuakeWorld supports two types of filtering: IP packet filtering and user id filtering. Userid filtering is the most convenient way to keep a specific person off of a server, but because anyone can create as many accounts as they want, a malicious user could just keep logging back in with a new account. If their ip address is banned, they won't be able to log on with any account from that computer. Unfortunately, most dialup accounts give a different ip address for each connection, so you may be forced to ban an entire subnet to keep a specific person off.

You can add or remove addresses from the filter list with:

addip < ip> removeip < ip>

The ip address is specified in dot format, and any unspecified digits will match any value, so you can specify an entire class C network with "addip 192.246.40".

Removeip will only remove an address specified exactly the same way. You cannot addip a subnet, then removeip a single host.

iplist Prints the current list of filters.

writeip Dumps "addip < ip> " commands to iplist.cfg so it can be execed at a later date. The filter lists are not saved and restored by default, because I beleive it would cause too much confusion.

filterban < 0 or 1>

If 1 (the default), then ip addresses matching the current list will be prohibited from entering the game. This is the default setting.

If 0, then only addresses matching the list will be allowed. This lets you easily set up a private game, or a game that only allows players from your local network.

2.8 The rendition 3d accelerated version of Quake looks very good. (Aug 22, 1996)

The image quality is significantly better than software - dithered, bilinear interpolated textures, and subpixel, subtexel polygon models.

It is faster than software even at 320*200, and at 512*384 it is almost twice as fast.

We do take a bit of a hit when we have to generate a lot of 16 bit surfaces, so occasionally the framerate gets a bit uneven, but overall it is a very solid improvement.

2.9 Aug 27, 1996

I haven't been working on QuakeWorld for the past week, because Michael has been working on the rendition port, which backburnered the win32 native port, which is a prerequisite for the QuakeWorld release.

Instead, I have been working on some better graphics code.

I now have a radiosity replacement for ligting quake maps. This will have a good effect on our Quake 2 maps, but probably won't be feasable for homebrew use.

I am finishing up a shadow volume pre-subdivision stage right now, and I need to do the resampling to quake light maps a bit better than I currently am, but I am very pleased with how it has turned out. A lot of the work will play into the next generation (NOT quake 2) game technology, for things like colored radiosity and dynamic shadows, but it is still cool just for calculating quake light maps.

Instead of placing lots of floating light entities, certain textures are tagged as being light emitters, so if you draw a flourecent light in the ceiling, it will automatically emit light from it's entire area. Lava emits light from the entire surface.

The simulation of the light is realistic now, instead of just hacked together. Light reflects off of surfaces, so ceilings will get lit even if there is no light pointing directly at them. The way light flows around corners looks eerily realistic.

I fixed the sky volumes so missiles dissapear inside them again, which was necessary to implement a new lighting feature: you can specify the position of the sun (or moon) in degrees, and the amount of light it will emit. The light then comes out completely parallel and at a constant intensity, passing through sky volumes to light up all outdoor areas in a consistant way. (this could be retrofited onto light.exe for people unable to run qrad)

Now for the bad news: a full size level takes several minutes to process on our quad alpha, and you MUST do a vis before running qrad (otherwise it would take about 1000 times longer). This wouldn't be out of reach even on mortal pentium systems (you can ask for a rough aproximation instead of the full job), except for the working set size. $>$ 100 megs. I'm NOT being wasteful here, either. I use a halved bit array for visibility interconnects, and sparse scaled shorts for energy transference.

I will release the source after we have been using it in production for a while, in case anyone wants to take a shot at reformulating it to be less memory intensive. It would be possible to get it down to 32 megs with some reduction in quality and a lot more wasted time, but then the run time would be obnoxious. I expect homebrew levels are going to have to stick with light.exe.

I am looking forward to doing a little more work on the other utilities as well. I want to break qbsp up into three seperate programs, so it will be able to run on machines with $<$ 32 megs of memory. Light can be sped up significantly if the PVS information is taken advantage of the way I did in qrad. I also have some theories on changing the algorithmic order of full vis processing time that I want to look into, especially now that the vis stage is put before production lighting.

Chapter 3

September

3.1 Sep 03, 1996

I have been cleaning up the utilities for another release (soon) where everything is in Visual C++ project format. All of the common code is now included from the same place, so adding support for pcx or 3d studio files will only require changes in one file now for all utilities.

I made a few simple changes to qcc, and it now compiles FOUR TIMES FASTER! A recompile on my pp200 only takes 5 seconds now. I knew the linear search for defs was hurting, but if I had realized it was that dominant (and easy to fix) I would have fixed it a long time ago. All I had to do was move defs to the head of the search list each time they were looked for, so they would be found fast the next time. A big hash table might be somewhat faster still, but this was trivial to add.

Qbsp now uses floats instead of doubles by default, which makes it a good deal less of a memory pig, but it seems to give numerical accuracy problems in some cases.

I am splitting qbsp up into three seperate programs, which will make the memory requirements much smaller than they are now, and will help me be more numerically rigorous. This is going to take some time, so it won't be in the next code release.

3.2 Ok, I'm back on QuakeWorld. (Sep 10, 1996)

While working on the new chat console, I fixed up a few little things with the regular console:
The pgup/pgdn keys now autorepeat.
The display does not pop down to the bottom when a new line is printed. If you aren't at the bottom, a few ^^^are printed on the last line.

There is now a message level you can set to cut down on the number of messages you get from the server (it saves a bit of bandwidth, too).
Level 0 gives everything.
Level 1 does not print item pickup messages.
Level 2 does not print player obituaries.
Level 3 does not print any messages from the server.
Level 4 does not print chat messages.

Player skin support is much nicer in QuakeWorld. There is a qw\skins directory, where you can just put a .pcx files for each skin. This will scale properly to hundreds of skins if you want to download them all.

I am still unsure of how much freedom I want to give with the skins.

One school of thought is that you just set the "skin" value to any name you want. If the other players don't have it, they fall back to the default.

The other school of thought is that skins are the restricted property of a specific clan, and only clan members get custom skins.

We have the ability to delegate specific information setting abilities to clan leaders. To join a clan, you have to get the leader to add the clan field to your user account. It is in the protected masterdata, so you can't edit it yourself.

3.3 There is a BETA distribution of 1.05 up on our ftp site. (Sep 13, 1996)

ftp://ftp.idsoftware.com/idstuff/unsup/q105beta.exe

It includes quake.exe winded.exe progs.dat, and a zip of the latest .qc files. Back up your current version before messing with this, in case something is broken.

There is new source code available. ftp://ftp.idsoftware.com//idstuff/source/qutils.zip has win-32 compiled versions and VC++ projects for all of the quake command line utilities. The qbsp/light/vis programs have not been tested all that well here, because we still use a dec alpha running unix for most of our work, but it has been converted from doubles to floats, which should reduce the memory requirements quite a bit (and possibly cause more numeric instabilities...).

I also uploaded the DOOM ipxsetup / sersetup source code, which we haven't had on our ftp site before.

We are going to begin internal testing of a 1.05 Quake release tomorrow. When we upload for general release, we will also upload the current .qc source. The old qcc.tar.gz file is still present on the ftp site for the original .qc files for now.

3.4 We are now beta testing the first release of QuakeWorld. (Sep 18, 1996)

It seems to be going well. The latency reductions are very real, and significantly improve control and responsiveness. There is some loss of smoothness in some cases, but you can tune the tradeoff to your liking.

I don't want everyone to think that it will instantly turn their 28.8 modem into a lan connection, but it should be solidly better than what we have in all circumstances.

On an internet connection that provides a stable flow of packets at a reasonably predictable rate, the gameplay is very good. Single packet drops are not even noticable, but dropping several packets at once or having a large variability in latency can still mess up a game.

The really cool ISPs will run QuakeWorld servers right at their pops, which should give 28.8 modem players 200 ms or less pings, providing a nearly flawless game.

3.5 Sep 22, 1996

I had a good time at the New York event – it was fun to meet a lot of the web and clan guys, and there was some good fierce deathmatch action.

The QuakeWorld beta has revved a couple more times. When things work right, its great, but it seems like it only works right on about half the windows machines we try it on.

We have at least a week of beating on it before a public release.

Chapter 4

October

4.1 Oct 14, 1996

I haven't been able to work on QuakeWorld for a week, and am unlikely to for a couple more days.

I've been working with the utilities and editor most of the time, but there has been a lot of other stuff going on lately that has kept me from being very productive.

4.2 Oct 16, 1996

The latest windows drivers from rendition fix the lousy GUI performance I had been complaining about to several people. While it may not be state-of-the-art 2d acceleration, it doesn't get in your way anymore. I do not consider windows performance to be a negative for rendition anymore.

Chapter 5

November

5.1 Nov 06, 1996

Over the weekend, I ported Quake to open-gl / win32. It is a complete port with all features supported. Its not really practical right now on most consumer level equipment, but it runs ok on my intergraph glz-13t. It will work on the 3dlabs cards with their next driver release, but it really isn't very optimized yet, so don't expect anything really fast.

If you have high end hardware (intergraph realizm or dec powerstorm), it is pretty cool.

It is interesting to look at quake with hires 24 bit trilinear texture mapping. Sometimes you just don't even notice anything is different, but sometimes you can just stare at a scene for quite a while apreciating it.

Throwing bits and pixels at a static design certainly doesn't hurt, but it really doesn't bring it to a new level. I think I know where the next level is, but I'm just at the very beginning of the work on the next game architecture.

We have three mostly-done things to release: winquake, glquake, and quakeworld. No, I don't know when any of them are actually going to be released. Sorry.

QuakeWorld will actually download a new level for you now if you want to wait for it. Little models and sounds are a lot more practical to customize a server with, though.

Today I got cornered by the three level designers and an artist and they jumped all over me for working on new research instead of polishing the quake 2 tools. I guess the rest of my week is spoken for...

5.2 Nov 23, 1996

QuakeWorld: there is one physics problem that is preventing me from releasing: you get stuck sometimes in moving plats. I am having a hard time tracking it down, and I only get to work on it sporadically. I will spend a bit more time this weekend.

I have given the source for the master server to the QSpy guys and Disruptor to do with as they will (rankings, queries, etc). I know I won't have the time to support it like it deserves.

WinQuake: Michael is almost done.... We only have one reported serious problem, but there are still a list of little issues. We are finally feeling stable on windows. Not everyone may be able to get optimal settings, but SOMETHING should work almost everywhere.

I have an NT alpha machine on order, so we will be compiling for that as well. We aren't going to be able to spend time writing assembly language for it like the intel version has, but it should still run at a pretty decent clip. IBM was talking about getting us a power-PC machine, but I haven't heard back from them in a while. If one shows up, we will compile for that as well.

GLQuake: 3DFX is writing a mini-gl driver that will run glquake. We exepect very high performance. 3Dlabs is also working with me on improving some aspects of their open-GL performance. DynamicPictures sent me an Oxygen board, but glquake did a horrible crash and burn with their beta drivers. They are looking into it. DEC is sending me a powerstorm to work with, which should perform in the same ballpark as the intergraph realizm I did the development work on.

I do expect the 3dfx version to outperform the $5k+ professional gl cards, due to its higher fill rate and the fact that they can tune directly for quake. There are certainly valid reasons to buy $5k cards (24+ bit z buffers, 24 bit color, 1280 res, etc), but don't cry when a $300 card eats their lunch :-).

Because of the very fill-rate intensive nature of the way I implemented this version of glquake (using a blend pass to do the light maps), performance is not likely to be very good on any current generation consumer level board except 3dfx. The next generation of most vendors cards should have sufficient fill rate, if claims are to be believed. I may do a surface cached version just for the experience, in any case.

One little public rant: if you are a 3d chip designer, don't even THINK about not supporting true alpha blending. Screen door transparency is not a valid replacement. (this means YOU, matrox!)

IRIX-GLQuake: In two weeks, Ed Hutchins from SGI is coming down and we are going to port glQuake to irix. This will only run (reasonably) on systems with hardware texture mapping: O2, impact, RE2, and IR. No indys, extremes, etc. You could probably use them as dedicated servers, though.

D3DQuake: I started bringing up glquake on direct-3d a couple days ago. We are going to have a very nice, apples-to-apples api comparison here. It takes four times as much code to do anything in D3D as it does in OpenGL, but I will tune both to the best of my abilities and we shall see if there is any performance advantage to either API. We should be able to do comparisons on both 3DFX and 3DLabs hardware with both apis.

Quake utilities: I have split qbsp into two programs: qcsg and qbsp2. The processing is more robust, faster, and more memory frugal, but it currently generates rathar unoptimized maps. I will be writing a seperate map optimizer program soon to get the counts and size back to what they should be. I will probably do a new tool release after we get some more testing time on them. All of the utilities except qbsp now automatically use as many processors as you have if you are running on NT.

New stuff: The outline of the next generation game engine is beginning to take a bit of shape... I'm applying the lessons learned from quake and I have a long list of new things to try. I want to polish off all the outstanding

quake stuff, then just go into hermit mode and work on the future while the other guys do Quake 2.

Chapter 6

December

6.1 I need to clear up a little bit of really flagrant misinformation here (Dec 13, 1996)

Digital Sight and Sound, a dallas based system integrator, run a little blurb in their customer update newsletter that seriously misrepresents my experiences with intergraph and sgi hardware.

The title was ”Z-13 versus sgi maximum impact, Z-13 wins” and it included statements like ”After two weeks of grueling tests, the z-13 finished 2.5 times faster overall than the Maximum IMPCT... and at less than 1/2 the price!!!”. This just isn't true, in saveral ways.

The last time I did side by side system testing involving SGI was pitting an SGI high impact against a intergraph GLZ1T. For heavy texture mapping applications like I was doing, the SGI high impact was three times faster than the intergraph. A realizm-13t is 2.5 times faster than the older GLZ1T, but a max impact is twice as fast as a high impact, so intergraph's best is still only half the speed of sgi's best workstation.

For the record, here is my impression of the 3D hardware I have worked with, unedited. Prices are for configurations as we would use them, not minimum.

Rendition Verite: ($150) It won't set any speed records, but it is an excellent value, the quake port is pretty good, and the risc architecture allows a lot of room for optimization.

3DFX: ($300) The highest textured fill rate of anything short of high end SGI hardware. 50 mpix, almost irrespective of options. You can't run a desktop or high res display on it, but at 640*480 it will stomp any other consumer level board, given equal optimization effort.

3dlabs permedia: ($300) Right now, this is the only well supported low end open GL card. and a good all around video card. The visual quality is hurt over most other 3d boards because the internal paths apear to be 15 bits, so you don't get dithered 24 bit color out of your blending operations. The fill rate is very sensitive to what options (z buffer, blending, color depth, etc) you have turned on. It peaks at over 40 mpix, but that is with no zbuffering and 4 bit textures. Realistic cases (16 bit textures, z buffering, perspective, bilinear) bring you down to around 10 mpix. There are some performance hangups in the drivers for heavy texturing applications, but 3dlabs is working on improving them.

The remaining boards are pretty much for professional use, not gaming.

Dynamic Pictures Oxygen: ($1500) I can't reccomend this to anyone doing texture mapping. It was slow and had rasterization errors.

3dlabs glint-TX: ($1000 - $3000) A reletively low fill rate (15 mpix peak, often a bit lower), but fairly stable in performance. Good GL drivers. Capable of supporting very high resolutions and amounts of texture memory. Available from several vendors at various (professional) price points.

Intergraph intense-3d: ($2000 - $5000). The fill rate still isn't very good (15 mpix), but it hardly cares when you turn on trilinear and blending. They rape you on the texture memory prices. I have gotten better performance (but not tons better) out of this than any glint card I have tested, but I have yet to use one of the dual TX boards. They are also incompatable with a lot of motherboards.

Intergraph realizm: ($7000 - $12000) The best graphics card you can buy for NT. As far as I know, you can't even get these seperately, you have to buy them as part of an intergraph system. The systems we use list

around $30k each. The fill rate is 33 mpix, and very insensitive to options. You can add an optional geometry board to get very high transform/light throughput. DEC resells these as their Powerstorm T series cards for alpha systems.

SGI O2: ($10000, full system) The fill rate was better than intergraph's high end, but the triangle throughput was lower. I expect SGI will be able to optimize the performance more in the coming months. It's not a knock-your-socks-off performer, but a real, fully configured system costs less than just the video option on a high end intergraph.

SGI IMPACT: ($25000 - $50000, full system) excellent fill rate and geometry rate. A very balanced, high performace system. The texture memories are very small (one or four megs), so it would probably be possible to contruct a pathalogical case that probably puts an intergraph ahead of it, but you would have to stretch. The prices are higher than intergraph, but only by about 30%.

SGI Infinite reality: ($100000+) Fill rate from hell. Polygons from hell. If you don't trip up on state changes, nothing will come within shouting distance of this system. You would expect that.

Our decision to go intergraph/NT over SGI/IRIX had a lot more to do with the NT vs IRIX part of it. I wish I could buy SGI hardware for NT. The last two SGI systems I did serious work on were messes of instability, but the O2 we just got last week does actually seem stable.

As far as the "which video card should I buy?" question from gamers, I'm afraid I can't give you a definitive answer. All of the first generation boards have some problem or another. The second generation, coming out within the next six months or so, will be more solidly positive.

6.2 QuakeWorld went up, got to around 4000 users, then the master server exploded. (Dec 17, 1996)

Disrupter and cohorts are working on more robust code now.

If anyone did it on purpose, how about letting us know... (It wasn't all the people that tried %s as a name)

6.3 Wow. Apple bought NeXT. That really brightened my day. (Dec 21, 1996)

I haven't generally been unhappy developing on NT, but I had been carrying a bit of sadness over several of the things we left behind when we moved from NEXTSTEP.

I wouldn't touch a mac for development right now, but if apple does The Right Thing with NeXT...

6.4 OpenGL vs Direct-3D (Dec 23, 1996)

I am going to use this installment of my .plan file to get up on a soapbox about an important issue to me: 3D API. I get asked for my opinions about this often enough that it is time I just made a public statement. So here it is, my current position as of december '96...

While the rest of Id works on Quake 2, most of my effort is now focused on developing the next generation of game technology. This new generation of technology will be used by Id and other companies all the way through the year 2000, so there are some very important long term decisions to be made.

There are two viable contenders for low level 3D programming on win32: Direct-3D Immediate Mode, the new, designed for games API, and OpenGL, the workstation graphics API originally developed by SGI. They are both supported by microsoft, but D3D has been evangelized as the one true solution for games.

I have been using OpenGL for about six months now, and I have been very impressed by the design of the API, and especially it's ease of use. A month ago, I ported quake to OpenGL. It was an extremely pleasant

experience. It didn't take long, the code was clean and simple, and it gave me a great testbed to rapidly try out new research ideas.

I started porting glquake to Direct-3D IM with the intent of learning the api and doing a fair comparison.

Well, I have learned enough about it. I'm not going to finish the port. I have better things to do with my time.

I am hoping that the vendors shipping second generation cards in the coming year can be convinced to support OpenGL. If this doesn't happen early on and there are capable cards that glquake does not run on, then I apologize, but I am taking a little stand in my little corner of the world with the hope of having some small influence on things that are going to effect us for many years to come.

Direct-3D IM is a horribly broken API. It inflicts great pain and suffering on the programmers using it, without returning any significant advantages. I don't think there is ANY market segment that D3D is apropriate for, OpenGL seems to work just fine for everything from quake to softimage. There is no good technical reason for the existance of D3D.

I'm sure D3D will suck less with each forthcoming version, but this is an oportunity to just bypass dragging the entire development community through the messy evolution of an ill-birthed API.

Best case: Microsoft integrates OpenGL with direct-x (probably calling it Direct-GL or something), ports D3D retained mode on top of GL, and tells everyone to forget they every heard of D3D immediate mode. Programmers have one good api, vendors have one driver to write, and the world is a better place.

To elaborate a bit:

"OpenGL" is either OpenGL 1.1 or OpenGL 1.0 with the common extensions. Raw OpenGL 1.0 has several holes in functionality.

"D3D" is Direct-3D Immediate Mode. D3D retained mode is a seperate issue. Retained mode has very valid reasons for existance. It is a good thing to have an api that lets you just load in model files and fly around without sweating the polygon details. Retained mode is going to be used

by at least ten times as many programmers as immediate mode. On the other hand, the world class applications that really step to new levels are going to be done in an immediate mode graphics API. D3D-RM doesn't even really have to be tied to D3D-IM. It could be implemented to emit OpenGL code instead.

I don't particularly care about the software only implementations of either D3D or OpenGL. I haven't done serious research here, but I think D3D has a real edge, because it was originally designed for software rendering and much optimization effort has been focused there. COSMO GL is attempting to compete there, but I feel the effort is misguided. Software rasterizers will still exist to support the lowest common denominator, but soon all game development will be targeted at hardware rasterization, so that's where effort should be focused.

The primary importance of a 3D API to game developers is as an interface to the wide variety of 3D hardware that is emerging. If there was one compatable line of hardware that did what we wanted and covered 90+ percent of the target market, I wouldn't even want a 3D API for production use, I would be writing straight to the metal, just like I allways have with pure software schemes. I would still want a 3D API for research and tool development, but it wouldn't matter if it wasn't a mainstream solution.

Because I am expecting the 3D accelerator market to be fairly fragmented for the forseeable future, I need an API to write to, with individual drivers for each brand of hardware. OpenGL has been maturing in the workstation market for many years now, allways with a hardware focus. We have exisiting proof that it scales just great from a $300 permedia card all the way to a $250,000 loaded infinite reality system.

All of the game oriented PC 3D hardware basically came into existance in the last year. Because of the frantic nature of the PC world, we may be getting stuck with a first guess API and driver model which isn't all that good.

The things that matter with an API are: functionality, performance, driver coverage, and ease of use.

Both APIs cover the important functionality. There shouldn't be any real

argument about that. GL supports some additional esoteric features that I am unlikely to use (or are unlikely to be supported by hardware – same effect). D3D actually has a couple nice features that I would like to see moved to GL (specular blend at each vertex, color key transparancy, and no clipping hints), which brings up the extensions issue. GL can be extended by the driver, but because D3D imposes a layer between the driver and the API, microsoft is the only one that can extend D3D.

My conclusion about performance is that there is not going to be any significant performance difference ($<$ 10%) between properly written OpenGL and D3D drivers for several years at least. There are some arguments that gl will scale better to very high end hardware because it doesn't need to build any intermediate structures, but you could use tiny sub cache sized execute buffers in d3d and acheive reasonably similar results (or build complex hardware just to suit D3D – ack!). There are also arguments from the other side that the vertex pools in d3d will save work on geometry bound applications, but you can do the same thing with vertex arrays in GL.

Currently, there are more drivers avaialble for D3D than OpenGL on the consumer level boards. I hope we can change this. A serious problem is that there are no D3D conformance tests, and the documentation is very poor, so the existing drivers aren't exactly uniform in their functionality. OpenGL has an established set of conformance tests, so there is no argument about exactly how things are supposed to work. OpenGL offers two levels of drivers that can be written: mini client drivers and installable client drivers. A MCD is a simple, robust exporting of hardware rasterization capabilities. An ICD is basically a full replacement for the API that lets hardware accelerate or extend any piece of GL without any overhead.

The overriding reason why GL is so much better than D3D has to do with ease of use. GL is easy to use and fun to experiment with. D3D is not (ahem). You can make sample GL programs with a single page of code. I think D3D has managed to make the worst possible interface choice at every oportunity. COM. Expandable structs passed to functions. Execute buffers. Some of these choices were made so that the API would be able to gracefully expand in the future, but who cares about having an API that can grow if you have forced it to be painful to use now and forever after? Many things that are a single line of GL code require half a page of

D3D code to allocate a structure, set a size, fill something in, call a COM routine, then extract the result.

Ease of use is damn important. If you can program something in half the time, you can ship earlier or explore more aproaches. A clean, readable coding interface also makes it easier to find / prevent bugs.

GL's interface is procedural: You perform operations by calling gl functions to pass vertex data and specify primitives.

```
glBegin (GL_TRIANGLES);
glVertex (0,0,0);
glVertex (1,1,0);
glVertex (2,0,0);
glEnd ();
```

D3D's interface is by execute buffers: You build a structure containing vertex data and commands, and pass the entire thing with a single call. On the surface, this apears to be an efficiency improvement for D3D, because it gets rid of a lot of procedure call overhead. In reality, it is a gigantic pain-in-the-ass.

```
(psuedo code, and incomplete)
v = &buffer.vertexes[0];
v->x = 0; v->y = 0; v->z = 0;
v++;
v->x = 1; v->y = 1; v->z = 0;
v++;
v->x = 2; v->y = 0; v->z = 0;
c = &buffer.commands;
c->operation = DRAW_TRIANGLE;
c->vertexes[0] = 0;
c->vertexes[1] = 1;
c->vertexes[2] = 2;
IssueExecuteBuffer (buffer);
```

If I included the complete code to actually lock, build, and issue an execute buffer here, you would think I was choosing some pathologically slanted case to make D3D look bad.

You wouldn't actually make an execute buffer with a single triangle in it, or your performance would be dreadfull. The idea is to build up a large batch of commands so that you pass lots of work to D3D with a single procedure call.

A problem with that is that the optimal definition of "large" and "lots" varies depending on what hardware you are using, but instead of leaving that up to the driver, the application programmer has to know what is best for every hardware situation.

You can cover some of the messy work with macros, but that brings its own set of problems. The only way I can see to make D3D generally usable is to create your own procedural interface that buffers commands up into one or more execute buffers and flushes when needed. But why bother, when there is this other nifty procedural API allready there...

With OpenGL, you can get something working with simple, straightforward code, then if it is warranted, you can convert to display lists or vertex arrays for max performance (although the difference usually isn't that large). This is the right way of doing things – like converting your crucial functions to assembly language after doing all your development in C.

With D3D, you have to do everything the painful way from the beginning. Like writing a complete program in assembly language, taking many times longer, missing chances for algorithmic improvements, etc. And then finding out it doesn't even go faster.

I am going to be programming with a 3D API every day for many years to come. I want something that helps me, rather than gets in my way.

John Carmack
Id Software