JUNE 30, 2012

QUAKE 3 SOURCE CODE REVIEW: ARCHITECTURE (PART 1 OF 5) >>

Since I had one week before my next contract I decided to finish my "cycle of id". After Doom, Doom Iphone, Quake1, Quake2, Wolfenstein iPhone and Doom3 I decided to read the last codebase I did not review yet:

idTech3 the 3D engine that powers Quake III and Quake Live.

The engine is mostly an evolution of idTech2 but there are some interesting novelties. The key points can be summarized as follow:

- Part 2 : New dualcore renderer with material based shaders (built over OpenGL Fixed Pipeline).
- Part 3 : New Network model based on snapshots.
- Part 4 : New Virtual Machines playing an essential part in the engine, combining Quake1 portability/security with Quake2 speed.
- Part 5 : New Artificial Intelligence for the bots.

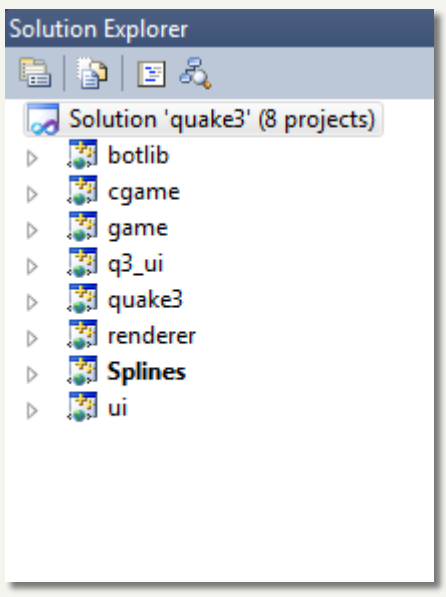
I was particularly impressed by :

- The virtual machines system and the associated toolchain that altogether account for 30% of the code released. Under this perspective idTech3 is a mini operating system providing system calls to three processes.
- The elegant network system based on snapshots and memory introspection.

As usual I wrote numerous notes that I have cleaned up and synthesized into drawings. I hope it will save time to some people but also encourage others to read more code and become better engineers.

Edit : Thanks for the support :) !

First contact



Since the venerable ftp.idsoftware.com was recently decommissioned the code can be found on id Software's GitHub account:

```
git clone https://github.com/id-Software/Quake-III-Arena.git
```

When it comes to comprehend a huge codebase I prefer to use XCode: SpotLight speed, Command-click to find definition and strings highlight make the tool more powerful than Visual Studio. But opening Quake III project showed that code rotting is not always about the code but also about the tools: XCode 4.0 is unable to open the Quake III XCode 2.0 projects.

In the end I used Visual Studio 2010 Professional on Windows 8: Upon installation of Visual Studio 2010 Productivity Power Tools the toolset was actually enjoyable.

The first striking thing is that the Visual Studio workspace is not made of one project but eight. Not all of them are used depending if the build is DEBUG or RELEASE (especially `game`,`cgame` and `q3_ui` : the virtual machines projects). Some of the projects are never used (`splines` and `ui`).

A table is better to summarize what project is contributing to which module:

Projects	Type	DEBUG Builds	RELEASE Builds	Comments
botlib	Static Library	botlib.lib	botlib.lib	A.I
cgame	Dynamic Libary/Bytecode	cgamex86.dll	-	
game	Dynamic Libary/Bytecode	qagamex86.dll	-	
q3_ui	Dynamic Libary/Bytecode	uix86.dll	-	
quake3	Executable	quake3.exe	quake3.exe	

renderer	Static Library	renderer.lib	renderer.lib	OpenGL based
Splines	Static Library	Splines.lib	Splines.lib	Used NOWHERE !
ui	Dynamic Libary/Bytecode	uix86_new.dll	-	Used for Quake III Arena.

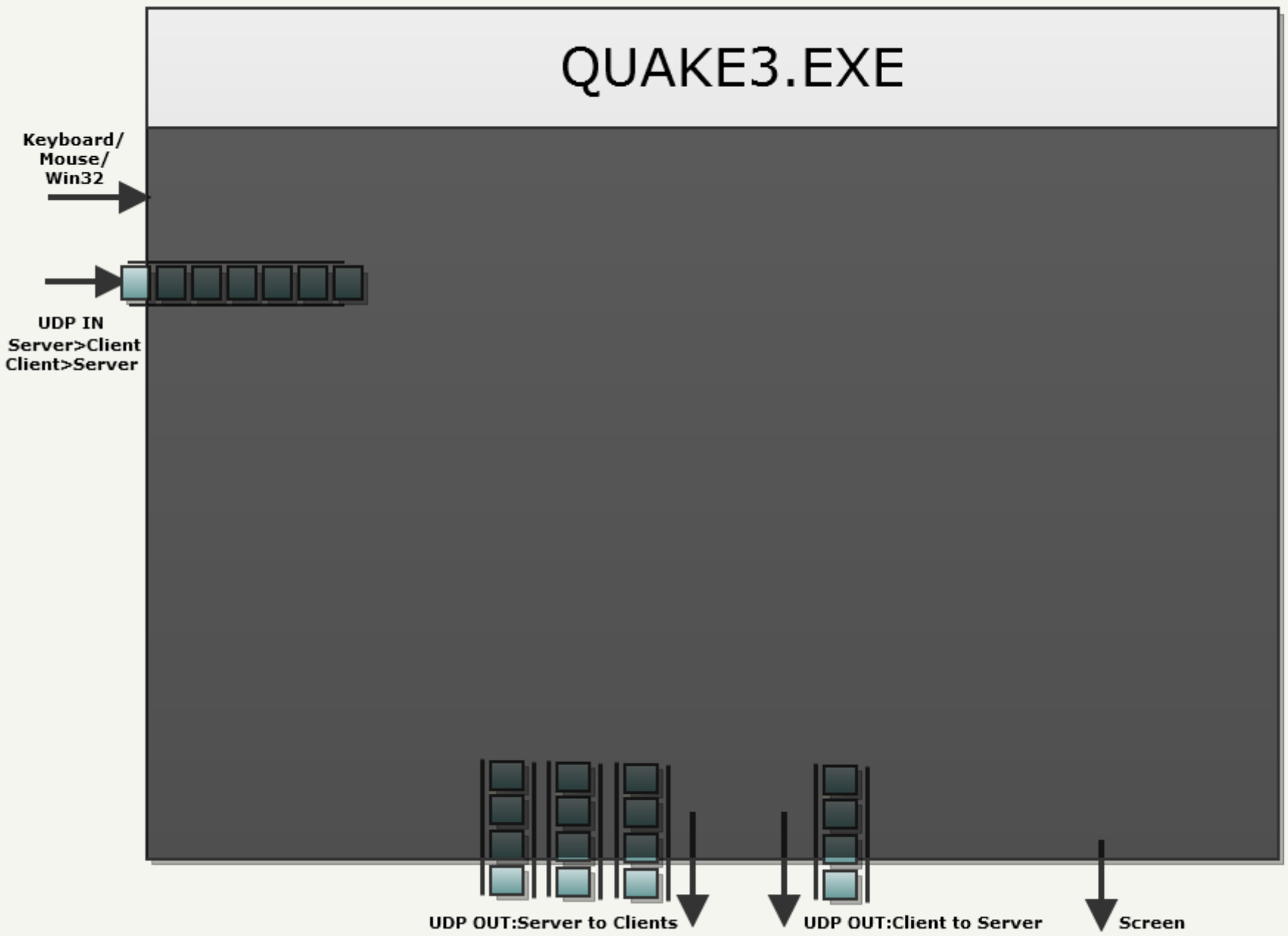
Trivia : idTech3 working title was "Trinity". Since idTech4 was called "Neo" I assumed it was from the "Matrix" franchise...but id Software stated in interview with firingsquad.com that it was named after "Trinity River in Dallas":

> **John :** I've got a couple of engine things that I'm working on, as far as research.
>
> **FS :** So is one of those things Trinity? I think there's been a little confusion about "Trinity."
>
> **John :** I was never really certain how this got as confusing as it did to everybody. After Quake, when I was starting on new rendering technologies and everything, everybody was just calling it "the next engine" or whatever. Michael Abrash suggested we just take Intel's tack of naming your next project after a river near you. We have the Trinity River in Dallas, and so it was just like "Trinity Engine," the next step.

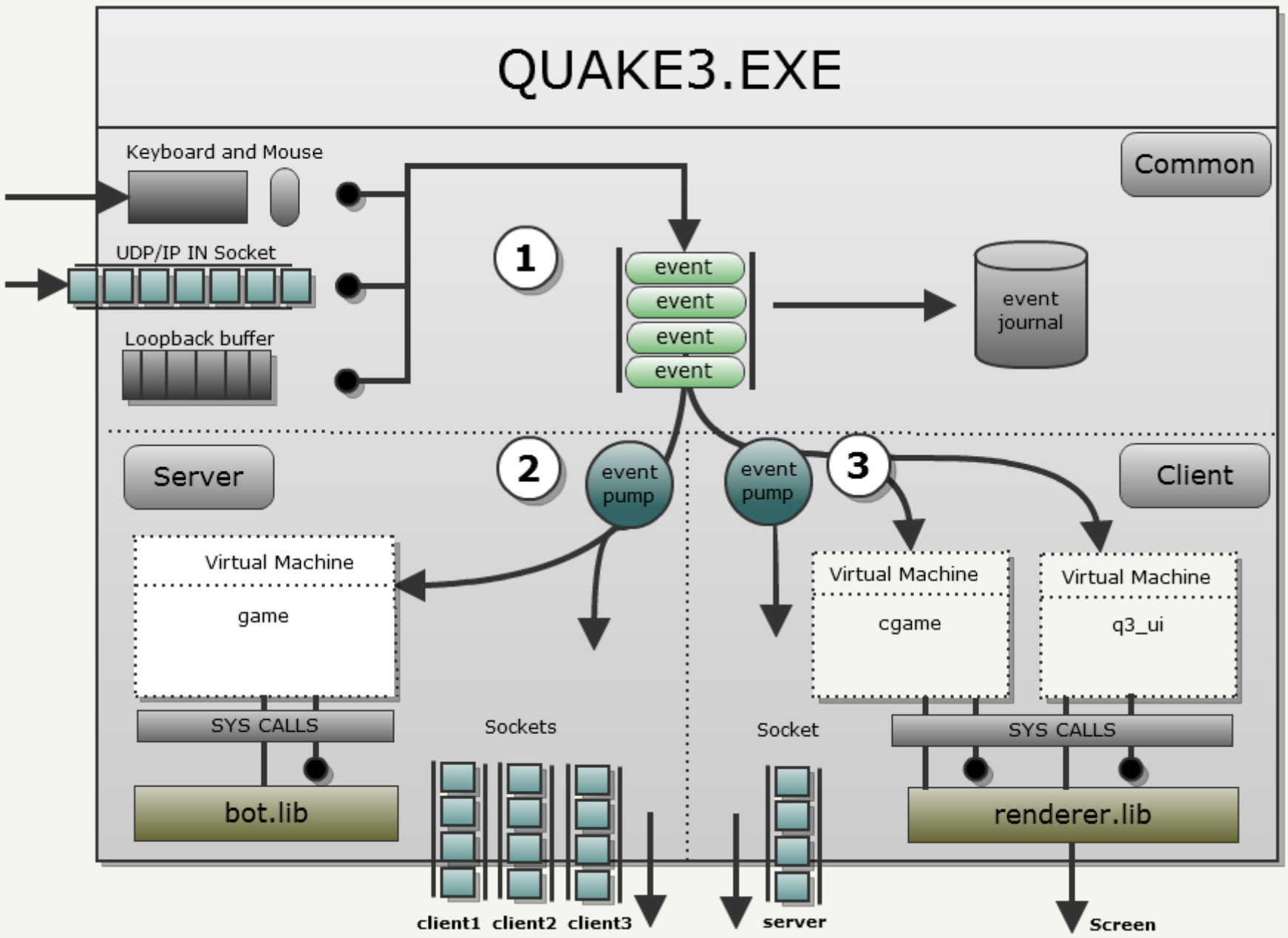
Edit (July 07, 2012) : Jeremiah Sypult contacted me after publication of this article: He "unrotted" the Mac OS X build with an XCode 4.0 project for Quake3. I have fixed it further and you can get it on here on github. You will need the Quake III Arena `baseq3` (not the demo version) and be sure to use the parameters `"+set vm_game 0 +set vm_cgame 0 +set vm_ui 0"` in order to use the dylib virtual machines. Open `/code/quake3.xcworkspace` and it builds in one click !!

Architecture

A convenient way to understand an architecture is to first look at the software as a black box receiving input (upper left arrows) and generating output (bottom arrows):



Then look how the inputs flows towards the outputs in a whitebox fashion with the 6 modules (`quake3.exe`, `renderer.lib`, `bot.lib`, `game`, `cgame` and `q3_ui`) interacting as follow:



Two important things to understand the design:

1. Every single input (keyboard, win32 message, mouse, UDP socket) is converted into an `event_t` and placed in a centralized event queue (`sysEvent_t eventQue[256]`). This allows among other things to record (journalize) each inputs in order to recreate bugs. This design decision was discussed at length in John Carmack's .plan on Oct 14, 1998.

2. Explicit split of Client and Server (this was outlined in a Q&A I did with John Carmack):

```
Fabien Sanglard: What do you think summarize the main innovations in idTech3 besides:
                  - Bot I.A
                  - Virtual Machine: Combining QuakeC portability and Quake2 dll speed.
                  - SMP & Shaders renderer.
                  - New Network code.

John Carmack: The explicit split of networking into a client presentation side and the server logical side was really the right
thing to do. We backed away from that in Doom 3 and through most of Rage, but we are migrating back towards it.  All of the Tech3
licensees were forced to do somewhat more work to achieve single player effects with the split architecture, but it turns out that
the enforced discipline really did have a worthwhile payoff.
```

   - The server side is responsible for maintaining the state of the game, determine what is needed by clients and propagate it over the network. It is statically linked against `bot.lib` which is a separate project because of its chaotic development history mentioned in page 275 of "Masters of Doom":

```
To make matters worse, a fundamental ingredient of the game - the bots - was missing. Bots
were characters controlled by the computer. A good bot would blend in with the action and flesh
out the scene like a robotic extra, as well as interact with the player. For Quake III, a deathmatch
only game, bots were essential for single-player action. They were implicitly complex because they
had to behave like human beings.

Carmack had decided, for the First time, to delegate the job of creating these bots to another
programmer in the company. But he failed to follow up. Once again, Carmack incorrectly assumed
that everyone was as self-motivated and adept as he was. He was wrong.

When Graeme struggled to rein in the work, it was discovered that the bots were completely ineffective.
They din't behave at all like human beings. They behaved, basically, like bots. The staff began to panic.
By March 1999, they had reason to be scared.
[..]

In the end the bot were farmed out to a well-known mod maker in the Netherlands, who heroically brought
them to life. (Note from Fab: This is Mr.Elusive: Jan Paul van Waveren).
```

   - The client side is responsible for predicting where entities are (latency compensation) and render the view. It is statically linked against `renderer` project: A separate project that would have allowed a Direct3D or even software renderer to be plugged in very easily.

The code

From a code point of view here is a partially unrolled loop that illustrate the event production and consumption by the client and server:

```
int WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{

    Com_Init

    NET_Init

    while( 1 )
    {
        // Common code
        IN_Frame()  // Add Win32 joystick and mouse inputs  as event_t to unified event queue.
        {
                IN_JoyMove
                IN_ActivateMouse
                IN_MouseMove
        }

        Com_Frame
        {
             // Common code
             Com_EventLoop    // Pump win32 message, UDP socket and console commands to the queue (sysEvent_t eventQue[256])
             Cbuf_Execute

             // Server code
             SV_Frame
             {
                 SV_BotFrame                                    // Jump in bot.lib
                 VM_Call( gvm, GAME_RUN_FRAME, svs.time )    // Jump in Game Virtual Machine where game logic is performed

                 SV_CheckTimeouts
                 SV_SendClientMessages                       // Send snapshot or delta snapshot to connected clients
             }

             // Common code
             Com_EventLoop
             Cbuf_Execute

             // Client code
             CL_Frame
             {
                 CL_SendCmd                                 // Pump the event queue and send commands to server.

                 SCR_UpdateScreen
                    VM_Call( cgvm, CG_DRAW_ACTIVE_FRAME);   // Send message to the Client Virtual Machine (do Predictions).
                         or
                    VM_Call( uivm, UI_DRAW_CONNECT_SCREEN); // If a menu is visible a message is sent to the UI Virtual Machine.

                 S_Update                                   // Update sound buffers
             }
         }

    }
}
```

Here is a fully unrolled loop that I used a a map while digging into the source code.

An interesting thing to notice here that perfectly illustrates how paramount the virtual machines are: Nowhere we see a call to `RE_RenderScene`: the function that performs culling and issue OpenGL commands. Instead what happen is:

1. Quake3.exe sends a message to the Client VM: `CG_DRAW_ACTIVE_FRAME` which signal that a refresh is needed.
2. The Virtual Machine performs some entity culling and prediction then call for OpenGL rendition via a Quake3 system call (`CG_R_RENDERSCENE`).
3. Quake3.exe receives the system call and actually calls `RE_RenderScene`.

```
 OpenGL                Quake3.exe                                Client Virtual machine
--------               ----------                               ----------------------
   |                        |                                           |
   |                        |                                           |
   |                  VM_Call(CG_DRAW_ACTIVE_FRAME)----------------->  CG_DrawActiveFrame
   |                        |                                          CG_DrawActive
   |                        |                                          trap_R_RenderScene
   |                        |                                          syscall( CG_R_RENDERSCENE, fd )
   |                  RE_RenderScene <-------------------------------------|
   |                        |                                           |
   | <----------------------|                                           |
   |                        |                                           |
```

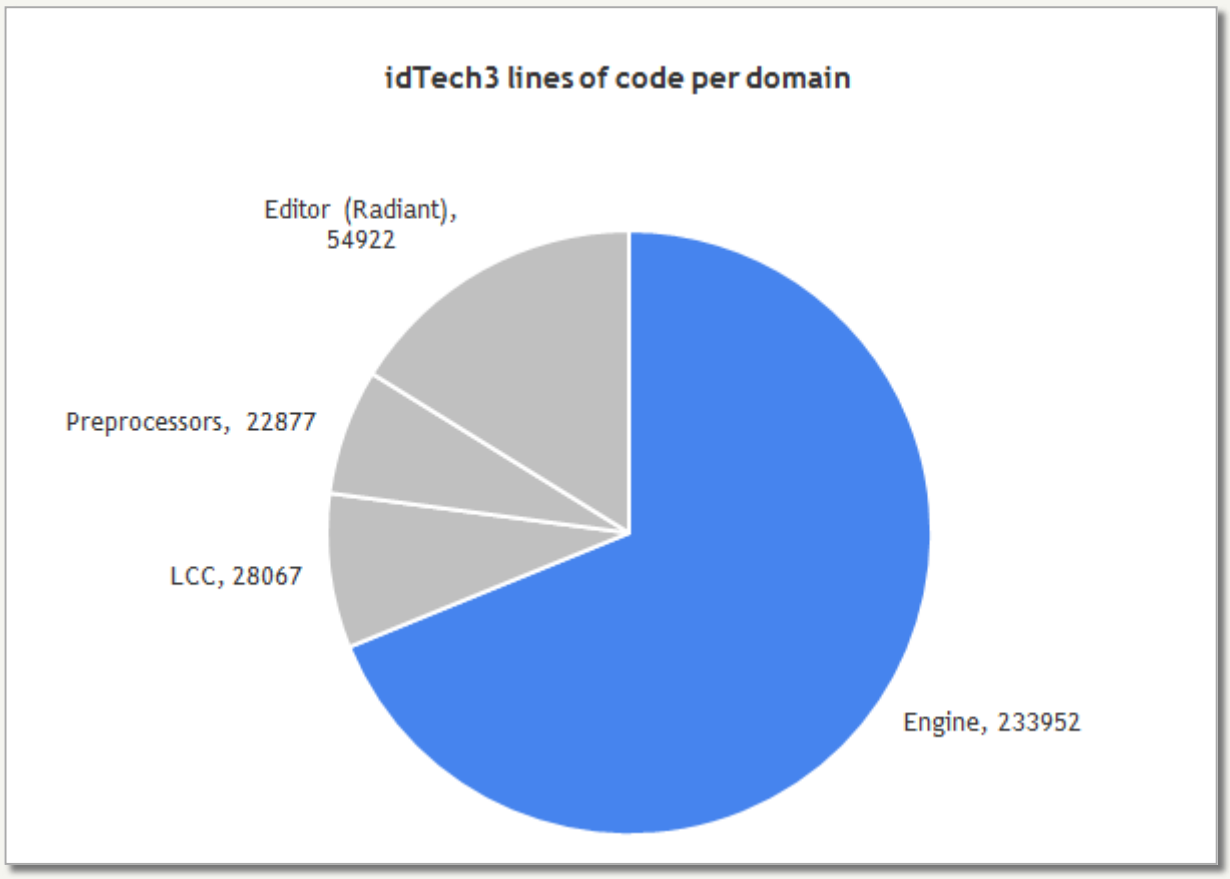
Statistics

Here are some stats from cloc:

```
-------------------------------------------------------------------------------------------------
                                          files          blank        comment           code
-------------------------------------------------------------------------------------------------
cloc-1.56.exe code common                   559          48630          73501         233952
cloc-1.56.exe lcc                           116           2270           1513          28067
cloc-1.56.exe q3asm q3map                    44           4987           5565          22877
cloc-1.56.exe q3radiant                     206          11870          13113          54922
-------------------------------------------------------------------------------------------------
```

```
TOTAL                                   919          68293          95509          341994
-------------------------------------------------------------------------------------------
```

On a pie chart we can vividly see how unusual the proportion are since 30% of the codebase is dedicated to tools:



This is explained partly because idtech3 features a ANSI C compiler: The open source Little C Compiler (LCC) is used to generate bytecode for the virtual machines.

Memory allocation

Two custom allocators at work here:

- Zone Allocator: Responsible for runtime,small and short-term memory allocations.
- Hunk Allocator: Responsible for on level load, big and long-term allocations from the pak files (geometry,map, textures, animations).

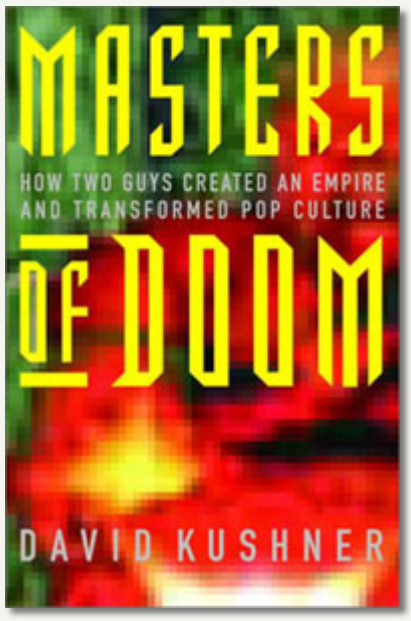
A Better tomorrow

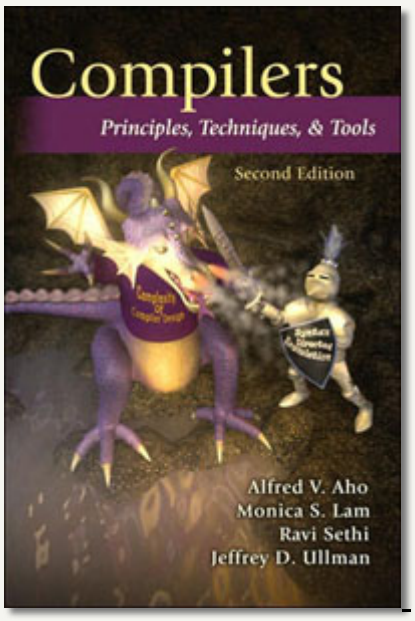
With this code review it seems I ran out of great 3D engine source code to read. But many great things are on the way in the CG world. I am especially looking forward the Occulus Rift VR kits:

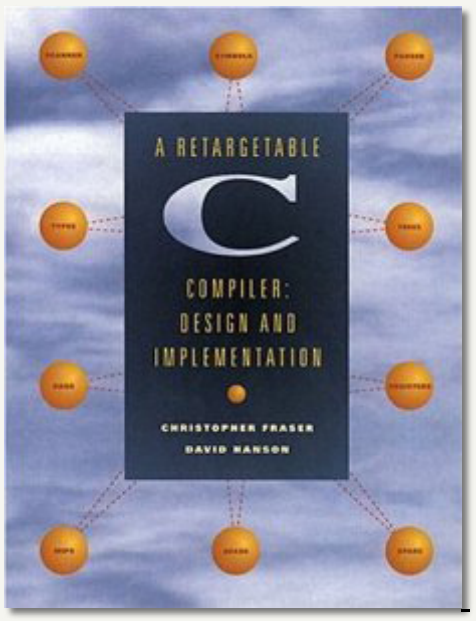
Building ,coding and understanding the testbed will be a lot of fun. It should arrive in late July so until then: Happy Hacking !

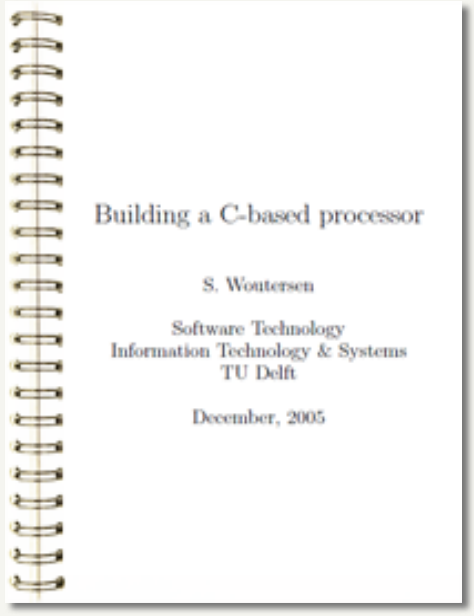
Recommended readings

Masters of Doom for history.
The two best compiler books to understand fully the Quake Virtual Machines.
A paper to understand LCC Intermediate Representation









Next part

The Rendition Model

@