MARCH 9TH, 2009

QUAKE ENGINE CODE REVIEW : ARCHITECTURE (1/4)

I happily dove into Quake World source code. Here is what I understood, hopefully it will help someone to swim.

This article is in four parts :

Quake Client

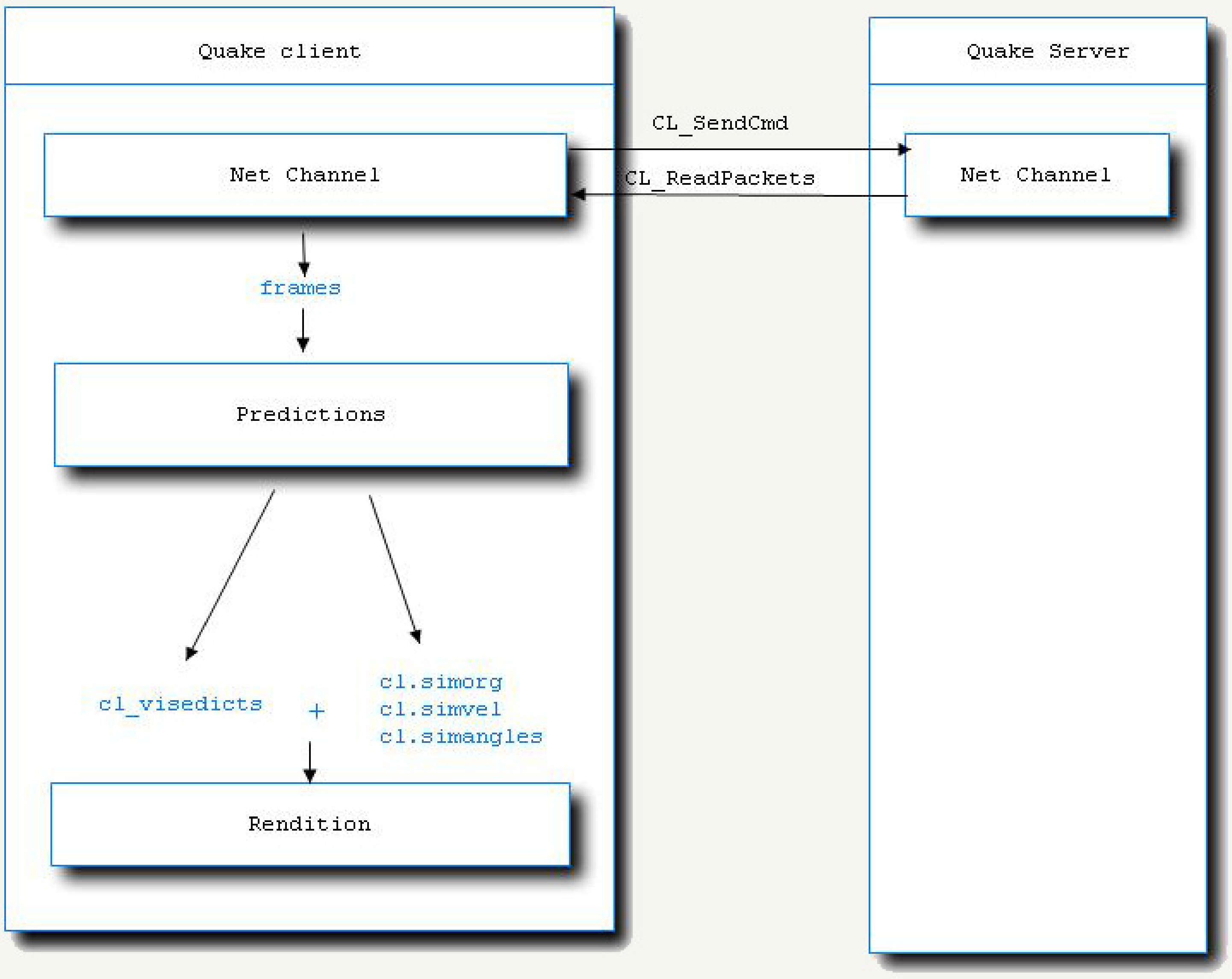
A good starting point to study Quake is the qwcl (client) project. The entry point WinMain can be found in **sys_win.c**. A quick summary of the code is as follows:

```
	WinMain
	{
		while (1)
		{
					newtime = Sys_DoubleTime ();
					time = newtime - oldtime;
					Host_Frame (time)
					{
						setjmp
						Sys_SendKeyEvents
						IN_Commands
						Cbuf_Execute

						/* Network */
						CL_ReadPackets
						CL_SendCmd

						/* Prediction//Collision */
						CL_SetUpPlayerPrediction(false)
						CL_PredictMove
						CL_SetUpPlayerPrediction(true)
						CL_EmitEntities

						/* Rendition */
						SCR_UpdateScreen
					}
					oldtime = newtime;
		}
	}
```

From here, we can identify the three key elements of Quake World:

- **Network** CL_ReadPackets and CL_SendCmd
- **Prediction** CL_SetUpPlayerPrediction, CL_PredictMove and CL_EmitEntities
- **Rendition** SCR_UpdateScreen

The **network** layer (also called Net Channel) outputs world information into the frames variable (An array of frame_t). They are picked up by the **prediction** layer, where collisions are also taken care of and data are outputted under the form of Visibility Edicts (cl_visedicts) and the POV definition. VisEdicts are used by the rendition layer, in addition to the POV (cl.sim*) variables to render the scene.



`setjmp` :

Setup a code waypoint, if anything bad happens, program jumps back here.

`Sys_SendKeyEvents` :

Retrieve Windows OS messages, minimizing etc... Update engine variable accordingly (world is not rendered if window is minimized for example).

`IN_Commands` :

Get joystick inputs.

`Cbuf_Execute` :

In every game loop, commands in the buffer are executed. Commands are generated mostly via the console, but can also come from the server or even direct keystroke.

The game starts with a `exec quake.rc` in the command buffer.

`CL_ReadPackets` and `CL_SendCmd` :

Take care of the **Network** piece of the engine. `CL_SendCmd` grabs the mouse/keyboard inputs, generates a command, which is then sent. As Quake World used UDP, the reliability is mainly replicated via a set of sequence/sequenceACK in the netChannel packet headers. Additionally, the last sent command is systematically re-sent. Regarding the flow control, there is no limitation on client side, updates are sent as fast as possible. On server side, a message is sent to a client only if a packet has been received and if the sending rate is below a "choke" limit. This limit is set from client side and sent to the server.

An entire section is dedicated to this part.

`CL_SetUpPlayerPrediction` , `CL_PredictMove` and `CL_EmitEntities` :

Take care of the **Prediction** piece of the engine as well as collisions. The goal is mainly to fight network communication latency.

An entire section is dedicated to this part.

`SCR_UpdateScreen` :

Take care of the **Rendition** piece of the engine. In this part, the BSP/PVS is extensively used. This is also where a fork occurs in the code based on `include`/`define`. Quake engine can render the world either with pure software or hardware accelerated.

An entire section is dedicated to this part.

Opening the zip and compiling

Opening the zip:

Upon opening q1sources.zip, there is two folders/Visual Studio projects: `QW` and `WinQuake`.

- `WinQuake` is the code with client and server code melted together, running within one single process (Ideally, they would have been two different processes if DOS had supported them). Network gaming was still possible via LAN only.
- `QW` is "Quake World" project, where Server and Client are meant to run on different machine (notice the client staring point is `WinMain` (in `sys_win.c`), whereas the Server stating point is `main` (also in `sys_win.c`)).

I studied Quake World, openGL rendered ; Upon opening the solution, 4 sub-projects can be seen:

- `gas2asm` - Utility to port assembler code from GNU ASM to x86 ASM
- `qwcl` - The client part of Quake
- `QWFwd` - Proxy sitting in front of Quake Servers
- `qwsv` - The server part of Quake

Compiling:

After installation of Windows and DirectX SDKs, compilation with Visual Studio 2008 raised one error :

```
            .\net_wins.c(178) : error C2072: '_errno' : initialization of a function
```

Nowadays, `_errno` is a Microsoft macro used for something else. You can fix these errors by changing the name of the variable from `_errno` to `qerrno` for example.

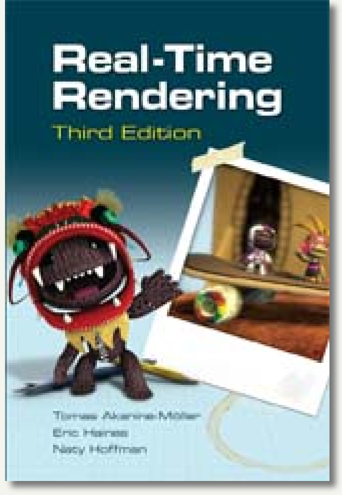
`net_wins.c`

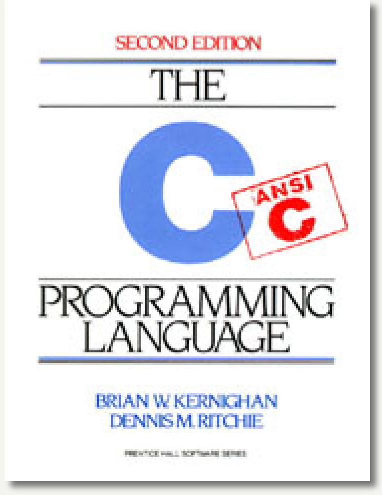
```
      if (ret == -1)
      {
            int qerrno = WSAGetLastError();

            if (qerrno == WSAEWOULDBLOCK)
                  return false;
            if (qerrno == WSAEMSGSIZE) {
                  Con_Printf ("Warning:  Oversize packet from %s\n",
                        NET_AdrToString (net_from));
                  return false;
            }


            Sys_Error ("NET_GetPacket: %s", strerror(qerrno));
      }
```

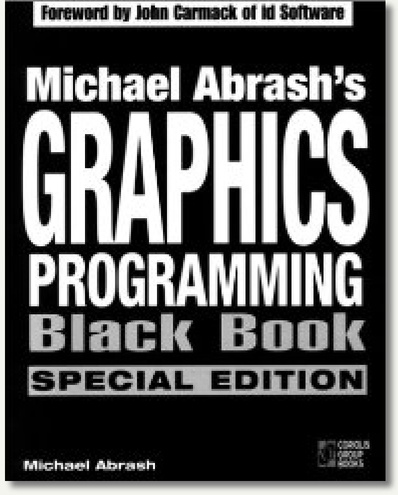
The linker will complain about LIBC.lib in qwcl project, just add it to the list of "Ignored Library", the 4 projects should build.

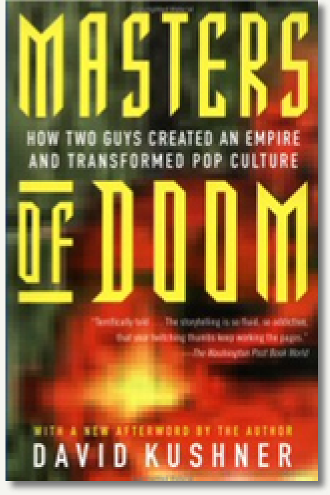
Tools

For the IDE, Visual Studio Express (free), was awesome.
A few books I highly recommend to read if you want to dig further into BSP/PVS based engine, Id Software and Quake:








MASTERS OF DOOM
HOW TWO GUYS CREATED AN EMPIRE AND TRANSFORMED POP CULTURE
WITH A NEW AFTERWORD BY THE AUTHOR
DAVID KUSHNER

My bookshelf during the Quake Source Code week: