FABIEN SANGLARD'S WEBSITE

HOME ABOUT FAQ EMAIL RSS TWITTER

FEBRUARY 14TH, 2013

DUKE NUKEM 3D: CODE LEGACY (PART 3 OF 4) >>

Build is an outstanding engine and the numerous games it powered brought very much deserved fame and fortune to both parties: Ken Silverman and 3D Realms.

Ken Silverman completed his contract: He delivered the binary of an astonishing 3D engine with well documented methods and assets formats. As a testimony of his skills, 3D Realms credited him with "Ken ' I can do that' Silverman".

But *Build* development was focused on features and performances, not portability or readability. Having read the code I believe open source developers stayed away from it because :

- It is discouragingly hard to read and extract knowledge from.
- It was not portable.

I have listed on this page a few of the difficulties I identified. I also released Chocolate Duke Nukem 3D a port that address those issues. Because I wanted people to remember what treasure of ingenuity it took to build a 3D engine back then. But also because I wanted people to understand how a teenager driven by passion contributed to one of the greatest game of all time.

Hard to understand

- Only three modules makes the entire codebase hard to mentally break down in sub-modules.
- `typedef` and `struct` are not used internally (only used for public data: `sectortype`, `walltype` and `spritetype`). Only arrays of primitive data types. (i.e: GRP Filesystem):

```
static long numgroupfiles = 0;
static long gnumfiles[MAXGROUPFILES];
static long groupfil[MAXGROUPFILES] = {-1,-1,-1,-1};
static long groupfilpos[MAXGROUPFILES];
static char *gfilelist[MAXGROUPFILES];
static long *gfileoffs[MAXGROUPFILES];

static char filegrp[MAXOPENFILES];
static long filepos[MAXOPENFILES];
static long filehan[MAXOPENFILES]
```

- Comments are sparce. This makes math methods difficult to understand (i.e: inside):

```
inside (long x, long y, short sectnum){
    walltype *wal;
    long i, x1, y1, x2, y2;
    unsigned long cnt;

    if ((sectnum < 0) || (sectnum >= numsectors)) return(-1);

    cnt = 0;
    wal = &wall[sector[sectnum].wallptr];
    i = sector[sectnum].wallnum;
    do{
        y1 = wal->y-y; y2 = wall[wal->point2].y-y;
        if ((y1^y2) < 0){
        x1 = wal->x-x; x2 = wall[wal->point2].x-x;
        if ((x1^x2) >= 0) cnt ^= x1; else cnt ^= (x1*y2-x2*y1)^y2;
        }
        wal++; i--;
    } while (i);
    return(cnt>>31);
}
```

- The drawing routines were x86 hand optimized assembly. Those were ported back to C later but there are still extremely hard to read as well.
- Symbols names are sometimes controversial:

```
static long xb1[MAXWALLSB], yb1[MAXWALLSB], xb2[MAXWALLSB], yb2[MAXWALLSB
static long rx1[MAXWALLSB], ry1[MAXWALLSB], rx2[MAXWALLSB], ry2[MAXWALLSB
static short p2[MAXWALLSB], thesector[MAXWALLSB], thewall[MAXWALLSB];
```

.

- Magic numbers (especially when manipulating walls, floor and ceiling) are mysterious.
- The main Translation Units (game.c and engine.c) are so big that they slow down IDE. Sometimes to the point of crashing XCode.

Hard to port

- Assumed Little-Endian CPU.
- Used `long` type across the engine expecting them to always be 32 bits wide.
- Assumed 32 bits addresses and manipulated them as integers.
- Features hand optimized x86 assembly with no C fallback.
- Expected 120 ticks per frame: The way DOS timer worked.
- No abstraction layer. Calls to filesystem/renderer were raw.

Lost ressources

Ken Silverman once kindly collaborated to the JonoF port and posted many explanation of his algorithms. Unfortunately the forum had to be taken down due to spam. It seems all those precious information are gone :(!

> *It is with disappointment that I've closed the forum on this site. Spam bots made it impossible to keep it under control, and although there is some valuable content within its posts, it's just not worth my or the other moderators' time to maintain it.*
>
> *I suggest you head over to the Duke4.net forums if you want to remain involved in the community.*

Comments

Fabien Sanglard @2013