FABIEN SANGLARD'S WEBSITE

HOME ABOUT FAQ EMAIL RSS TWITTER

FEBRUARY 14TH, 2013

DUKE NUKEM 3D CODE REVIEW: INTRODUCTION (PART 1 OF 4) >>

Since I left my job at Amazon I have spent a lot of time reading great source code. Having exhausted the insanely good idSoftware pool, the next thing to read was one of the greatest game of all time : Duke Nukem 3D and the engine powering it named "*Build*".

It turned out to be a difficult experience: The engine delivered great value and ranked high in terms of speed, stability and memory consumption but my enthousiasm met a source code controversial in terms of organization, best practices and comments/documentation. This reading session taught me a lot about code legacy and what helps a software live long.

As usual I cleaned up my notes into an article. I hope it will inspire some of us to read more source code and become better engineers.

I would like to thanks **Ken Silverman** for proof-reading this article: His patience and diligent replies to my emails were instrumental.

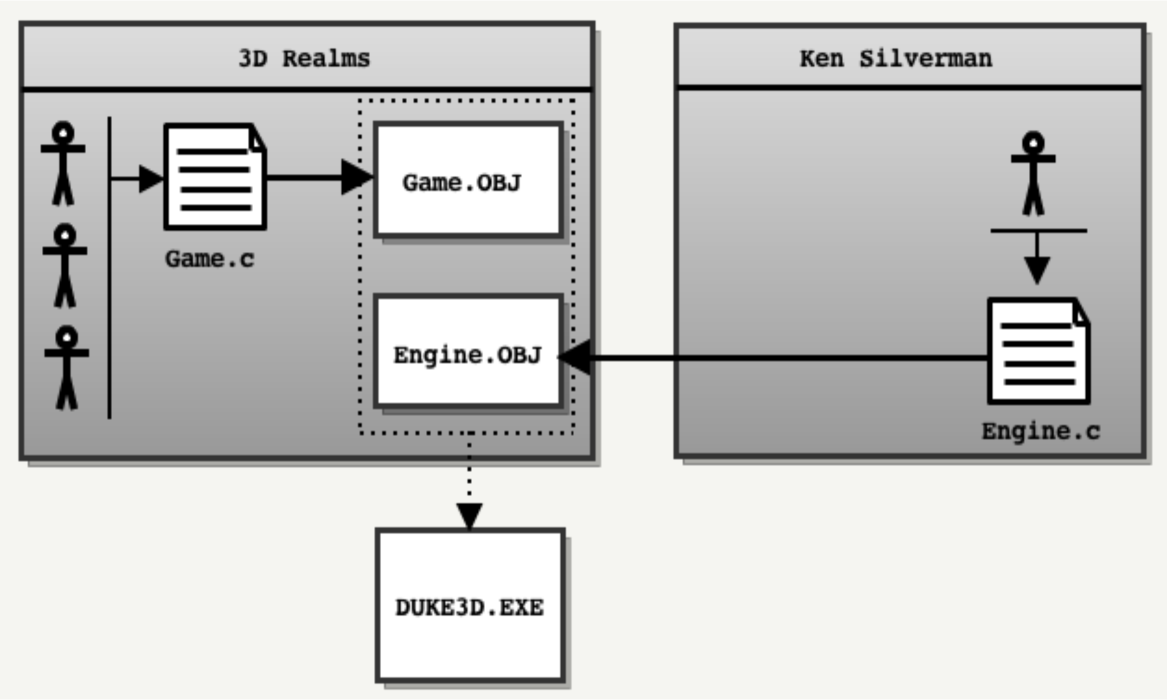
Genesis

Duke Nukem 3D is not one codebase but **two** :

- *Build* engine: Providing rendition, network, filesystem and caching services.
- Game Module: Using *Build*'s services in order to generate a gaming experience.

Why such a division? Because back in 1993, when development started, very few people had the skills and dedication to produce a good 3D engine. When 3D Realms decided to write a game that would challenge *Doom*, they had to find the technology that would power it. That is where Ken Silverman enters the picture.

According to his well-documented website and interview, Ken Silverman (18 years old at the time) wrote a 3D engine at home and sent a demo for evaluation to 3D Realms. They found his skills promising and worked out a deal:



Silverman would write a new engine for 3D Realms but he would keep the source code. He would only deliver a binary static library (Engine.OBJ) with an Engine.h header file. The 3D Realms team on their side would take care of developing the Game module (Game.OBJ) and would also release the final executable DUKE3D.EXE.

Unfortunately both part of the game were not open sourced simultaneously:

- The Engine module source code was released by Ken Silverman on June 20, 2000 .
- The Game module source code was released by 3D Realms on April 1, 2003.

As a result the full source code was available 7 years after the game was released.

Trivia : The name of the engine "*Build*" was chosen by Ken Silverman when creating the directory for the new engine: He used a thesaurus to search for a synonyms of "Construction".

First contact

Since the original code released rotted a long time ago (It was targeted to Watcom C/C++ compiler and DOS systems) I tried to find something like Chocolate doom: A port that accurately reproduces the experience of Duke Nukem 3D as it was played in the 90s and would flawlessly compile on modern systems.

It turned out the Duke Nukem open source community is not very active anymore: Most ports have rotted again, some have MacOS 9 PowerPC targets and the only one still maintained (EDuke32) has evolved too far from the original code. I ended up working with xDuke even though it did not compile on Linux or Mac OS X (which ruled out XCode: An outstanding IDE when it comes to reading code and profiling).

xDuke's Visual Studio Solution is fidele to the original workflow. It features two projects: Engine and Game: The "Engine" project compiles to a static library (`Engine.lib`) and the "Game" project (featuring the `main` method) links against it in order to generate a `duke3D.exe`.

Upon opening VS, the engine source felt unwelcoming with difficult filenames (`a.c`, `cache1d.c`). Opening those files reveals something hostile to the eyes and the mind.

An example among many others from Engine.c (line 693):

```
if ((globalorientation&0x10) > 0) globalx1 = -glob
if ((globalorientation&0x20) > 0) globalx2 = -glob
globalx1 <<= globalxshift; globaly1 <<= globalxshi
globalx2 <<= globalyshift;  globaly2 <<= globalysh
globalxpanning <<= globalxshift; globalypanning <<
globalxpanning += (((long)sec->ceilingxpanning)<<2
globalypanning += (((long)sec->ceilingypanning)<<2
globaly1 = (-globalx1-globaly1)*halfxdimen;
globalx2 = (globalx2-globaly2)*halfxdimen;
```

Note : If a file/variable name features a number: it is probably not a good name !

Trivia : The last part of `game.c` features a draft of Duke V scenario !

Note : xDuke port uses SDL but the cross-platform API advantage is lost to WIN32 timers (`QueryPerformanceFrequency`). It seems the SDL Timer used to be too innacurate in order to emulate DOS 120Hz tick rate.

Building

After setting up SDL and DirectX header/lib location the code will build in one click: That is very pleasant. The only last part is to get a `DUKE3D.GRP` asset file and the game runs....well kinda :/.... SDL seems to have some palette issues with Vista/Windows 7:

Running in Windowed Mode (or better, use Windows 8) seems to fix the issue:

Immersion

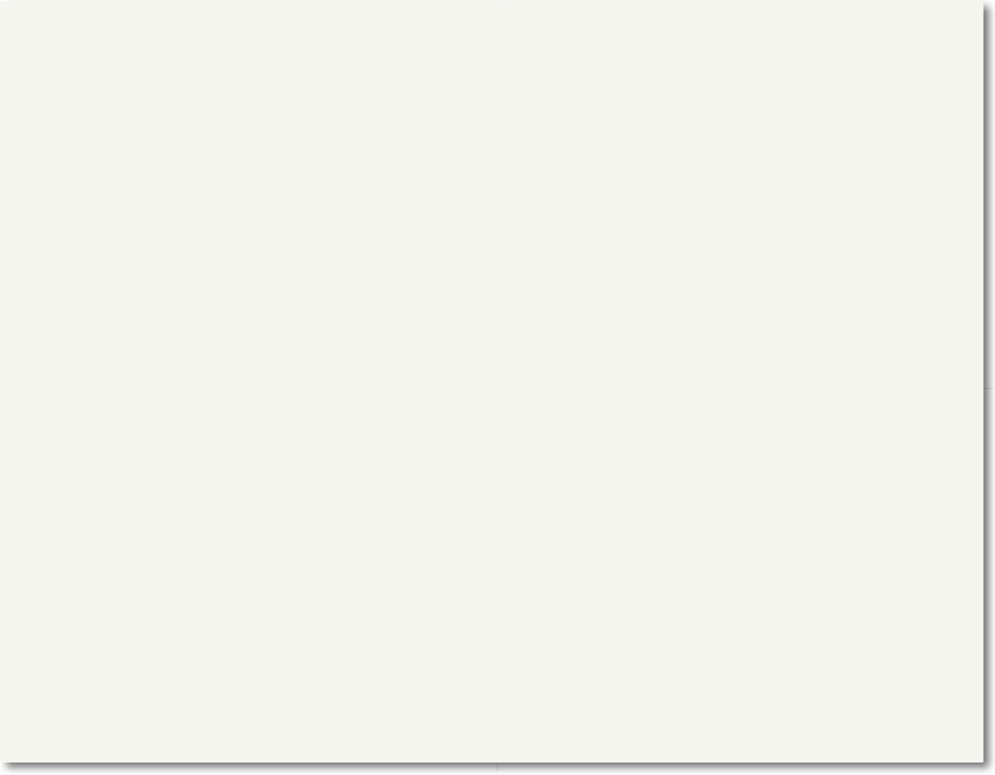
Now the game is running. Within the first seconds *Build* shines with, in order:

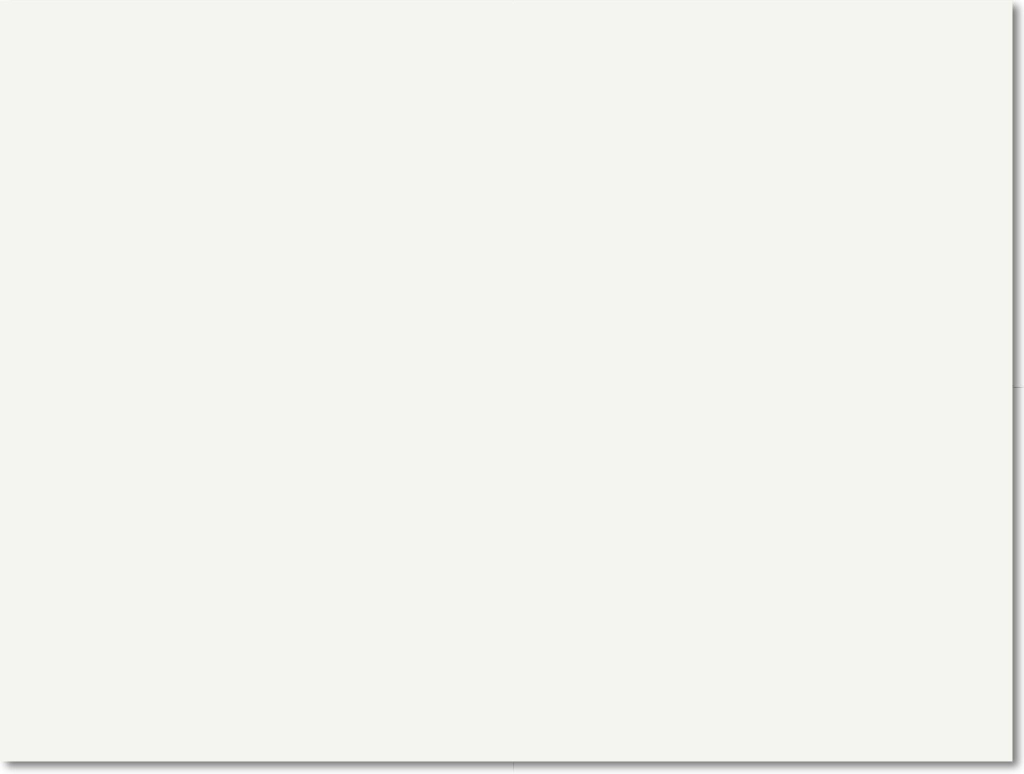
- Sloped floors.
- Realistic environments.
- Free fall.
- Feeling of true 3D.

The last point is probably what struck players the most in 1996. The level of immersion experienced was unprecedented. Even when the technology reached its limit because of 2D maps, Todd Replogle and Allen Blum implemented "sector effectors" allowing to teleport the player and improve the feeling of evolving in a three dimensional world. This feature is used in the legendary "L.A Meltdown" map:

When the player jumps in the ventilation shaft:

A sector effectors triggers and teleports the player at a totally different location on the map just before "landing" :





Trivia : Ever since, 3D engines have tried to improve immersion with better graphics. Things may take a new direction with the release of the Virtual Reality device "The Oculus Rift".

Great games age nicely and Duke Nukem is no exception: Twenty years later the game is still incredibly fun to play. Except now we can start digging in the source!

Engine Library Overview

The engine code is in one file of 8503 lines with 10 master functions (`Engine.c`) and two auxiliary files:

- `cache1.c`: Contains the virtual filesystem (sic) and the cache system routines.
- `a.c`: A C reverse-engineered implementation of what used to be highly optimized x86 assembly. It works but is a monstruous pain in the ass to read :(!

Three translation units and few functions makes the high level architecture hard to comprehend. Unfortunately this is not the only difficulties a reader will face.

As full page is dedicated to the engine: *Build* Engine Internals.

Game Module Overview

The game module is built completely on top of the Engine module, the way an operating system's system calls are used by a process: Anything the game does is done thought *Build* (drawing, loading assets, filesystem, caching system, ...). The only exception is the sounds/music system which *Game* completely owns.

Since I was mostly interested in the engine I did not read it in depth. But it shows more experience and organization: 15 files divide the source into clear modules. It even features a `types.h` (precursor of `stdint.h`) in order to improve portability.

A few interesting things :

- `game.c` is a beast featuring 11,026 lines of code.
- `menu.c`" features a 3000 lines "switch case".
- Most methods have "void" parameters and return "void". Everything goes via global variables.
- Methods naming does not use camelCase or NAMESPACE prefix.
- Features a neat parser/lexer even though token values are passed via decimal values instead of `#define`.

Overall this part code is easy to read and understand.

Source code legacy

Looking at the innumerable ports that spawned Doom/Quake, I was always puzzled to see so few ports of Duke Nukem 3D. The same question arose when the engine was ported to OpenGL only when Ken Silverman decided to do it himself.

Now that I have looked at the code I can risk an explanation in a dedicated page: The legacy of Duke Nukem 3D source code.

Chocolate Duke Nukem 3D

Admiring the engine and loving the game so much, I could not let things this way: I started Chocolate Duke Nukem 3D, a port of the Vanilla source code with two goals in mind:

- Education: Easy to read/understand and very portable.
- Fidelity: The gaming experience should be similar to what ran in 1996 on our 486s.

I hope this inntiative will help the code to find a new legacy. The key modifications are described in Chocolate Duke Nukem 3D page.

Recommended readings

None. Go rock climbing: It is awesome !
If really you insist, read id as Super-Ego: The Creation of Duke Nukem 3D

Comments

60 Comments **Fabien Sanglard's website** 1 **Login**

♡ **Recommend** 3 **Share** **Sort by Oldest**

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Justin Meiners • 4 years ago

Awesome Thank You!

^ | ˅ • Reply • Share ›

Gustavo • 4 years ago

Check out archive.org for the forum: http://web.archive.org/web/...

1 ^ | ˅ • Reply • Share ›

Lucas • 4 years ago

Unless I'm mistaken, JonoF's forum is backed up in the web archive here: http://web.archive.org/web/...

1 ^ | ˅ • Reply • Share ›

Jan Zwiener • 4 years ago

Another very interesting article, thank you Fabien!

^ | ˅ • Reply • Share ›

Ben. • 4 years ago

Thanks for this. I really appreciate people having a closer look on something and sharing their insights.

It's a pitty that they could not just refurbish the graphics, add some new maps and release a new Duke. What they released was a "let's get things done" game and called it "Duke Nukem Forever".

Maybe there will be a "Duke Nukem Community Edition" with all the cool gameplay...

Thanks for your work!

1 ^ | ˅ • Reply • Share ›