FEBRUARY 14TH, 2013

DUKE NUKEM 3D: CHOCOLATE DUKE NUKEM 3D (PART 4 OF 4) >>

Chocolate Duke Nukem 3D is a port of Duke Nukem 3D aimed at education. The main goal is to clarify the code so programmers can extract knowledge easily and get a better idea of what it was to program game engines in the 90s.

Like an archeologist working on bones it was important to keep things the way they were and only the "dust" has been removed with focus on:

- Readability : Make the code easy to understand.
- Portability : Make the code easy to compile, run and tinker with.

Binaries

This is a port for game developers that want to learn about the architecture and source code of Duke Nukem 3D. If you just want to play the game, I would recommend to use EDuke32 instead.

If you want to play Chocolate Duke Nuken 3D anyway, just download the source code which features an XCode/Visual Studio project and built it : Link to the source code.

Portability

The lack of portability was an issue now Chocolate Duke Nukem 3D compiles on Windows, Intel MacOS X and Linux is one `makefile` away. Here is what has been done:

- Usage of Integral type aliases now guarantee the size of integers. The `long` was used everywhere because it was tought during development that this type would always be 32 bits wide. It is one of the reason the engine cannot be compiled in 64 bits mode. Using `int32_t` from the standard `inttypes.h`.
- Removal of `char` for arithmetic operations: Since it can be `signed` or `unsigned` depending on the platform, `char` for maths resulted in nasty wraparound ; `char` should only be used for strings. For arithmetic, *Build* is now explicit with `int8_t` or `uint8_t` from `inttypes.h` that guaranty signedness.
- Removal of platform dependent API. Back when SDL timer accuracy was average, the port had trouble replicating the mandatory 120ticks/frame. Now the engine either use SDL or provide a platform specific implementation for POSIX and Windows.

The code is much more portable but still not 64 bits ready: More work is still necessary in the interface between the *Engine Module* and the *Drawing Module* where memory address are manipulated as 32 bits integers. This part will require many hours and I am unsure I will be able to dedicate that much time.

Understandability

Most of the workload went into making the code easy to read. Here is a list of what was done:

Modules definition

The vanilla source code was essentially contained in three translation units:

- `Engine.c` : Accounting for 95% of the code.
- `a.c` : Containing a crude C implementation of what was once optimized ASM.
- `cache1d.c` : Containing the caching and GRP file systems.

The code has been redistributed in units that give a clear idea of what the code inside does :

- Engine.c : Now 50% of the code.
- display.c : SDL surfaces buffers where the screen is rendered, palette utilities.
- draw.c : The C implementation of the ASM routines.
- tiles.c : The sprite engine.
- filesystem.c : Anything abstracting the GRP filesystem.
- network.c : Multiplayer is not here.
- cache.c : The custom memory allocator and cache service.
- math.c : Most of the fixed arithmetic helper functions are here.

I was tempted to break down Engine.c into a frontend and backend: Mimicking the Quake3/Doom3 architecture with two parts communicating via the bunch stack. In the end I judged it too far from the original spirit of the engine and dropped the idea.

Data structure

Build used struct to communicate with the *Game Module* via build.h but internally everything was done with arrays of primitive data types: No struct and no typedef. This has been modified and especially with regards to the Visual Surface Determination and Filesystem:

Before:

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

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

After:

```
// A typical GRP index entry:
//     - 12 bytes for filename
//     -  4 for filesize
typedef uint8_t  grpIndexEntry_t[16];

typedef struct grpArchive_s{
    int32_t  numFiles             ;//Number of files in the archive.
    grpIndexEntry_t  *gfilelist   ;//Array containing the filenames.
    int32_t  *fileOffsets         ;//Array containing the file offsets.
    int32_t  *filesizes           ;//Array containing the file offsets.
    int fileDescriptor            ;//The fd used for open,read operations.
    uint32_t crc32                ;//Hash to recognize GRP archives: Duke Share
} grpArchive_t;

//All GRP opened are in this structure
typedef struct grpSet_s{

   grpArchive_t archives[MAXGROUPFILES];
   int32_t num;

} grpSet_t;
```

Symbols name sanitization

Variable names have been modified when they provided little clue about their usage:

Before:

```
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];
```

After:

```
enum vector_index_e {VEC_X=0,VEC_Y=1};
enum screenSpaceCoo_index_e {VEC_COL=0,VEC_DIST=1};
typedef int32_t vector_t[2];
typedef int32_t coo2D_t[2];

// This is the structure emitted for each wall that is potentially visible.
// A stack of those is populated when the sectors are scanned.
```

```
typedef struct pvWall_s{
    vector_t cameraSpaceCoo[2]; //Camera space coordinates of the wall endpoint
    int16_t sectorId;           //The index of the sector this wall belongs to
    int16_t worldWallId;        //The index of the wall in the map database.
    coo2D_t screenSpaceCoo[2];  //Screen space coordinate of the wall endpoints
} pvWall_t;

// Potentially Visible walls are stored in this stack.
pvWall_t pvWalls[MAXWALLSB];
```

Comments and documentation

- Documentation : Since the JoFo forum posts are gone, I hope the Build Internals page will helps developers to have an idea of the high level architecutre of the engine.
- Comments : This is the point where I tried to invest most of the time. I am a huge believer of a lot of comments in code (Dmap is a great example of source with more comments than statements).

Magic numbers

I haven't had the time to remove all the magic numbers. Change decimal literal in favor of `enum` or `#define` would improve readability a lot.

Memory allocation

Chocolate Duke attemps to avoid global variables. Especially if they are used only for the lifetime of a frame. In those cases the memory used will be on the stack:

Before:

```
long globalzd, globalbufplc, globalyscale, globalorientation;
long globalx1, globaly1, globalx2, globaly2, globalx3, globaly3, globalzx;
long globalx, globaly, globalz;

static short sectorborder[256], sectorbordercnt;

static char tablesloaded = 0;
long pageoffset, ydim16, qsetmode = 0;
```

After:

```
/*
```

```
FCS:
Scan through sectors using portals (a portal is wall with a nextsector attribut
Flood is prevented if a portal does not face the POV.
*/
static void scansector (short sectnum)
{

    //The stack storing sectors to visit.
    short sectorsToVisit[256], numSectorsToVisit;
    .
    .
    .
}
```

Note : Be careful when using a stack frame to store big variables. The following code ran well when compiled on clang and gcc but failed with Visual Studio:

```
int32_t initgroupfile(const char  *filename)
{
    uint8_t          buf[16]                  ;
    int32_t          i, j, k                  ;
    grpArchive_t*    archive                  ;
    uint8_t        crcBuffer[ 1 << 20]    ;

    printf("Loading %s ...\n", filename)   ;
    .
    .
    .
}
```

A stack overflow occurred because Visual Studio reserves only 1MB for the Stack by default. Trying to use 1MB overflowed the stack and that made `chkstk` very unhappy. This code ran fine with Clang on Mac OS X.

Source code

The source code is available on github.

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 ›