Inside Wolfenstein

In general, as I promised... Everything about Wolf3D as it is. :)

Preamble

We will talk here about the family of games in the genre of FPS (First Person Shooter), developed by ID Software in 1991-92: **Wolfenstein 3D** (there are several versions, including shareware and commercial version) and **Spear Of Destiny** (there is a free demo version, and several "official" additions). Technically, all these games are quite similar, so from now on I will call them all simply "Wolfenstein". In cases where it is necessary to make a distinction, I will talk (respectively) about **Wolf** and **SOD**. In addition to these games from ID Software, there are several with similar technology that use an engine licensed from ID, but released by other developers: the **Blake Stone** family (Aliens of Gold, Planet Strike, ...), **Corridor 7**, etc. The most advanced "Wolfenstein-like" game, of course, should be considered **Rise Of The Triad** (**ROTT**) by Apogee. (Technically, this is the most serious of the alterations of the original game engine: all the code has been moved to a "flat" 32-bit model, added floor and ceiling texturing, partially transparent wall textures, several height levels, shading and fog effects, etc. - but, with all this, it is still based on the good old Wolfenstein.) Well, just in case, I will say that everything written below has nothing to do with modern games released under the Wolfenstein brand (they use other, somewhat more modern technologies :).

In general, the Wolf engine was very popular in the early 1990s - but when ID Software developed a revolutionary new engine for DOOM, Wolf quickly began to be forgotten. At the end of 1996, the source code for Wolf3D was published and is still freely available on the ID Software server at ftp://ftp.idsoftware.com/idstuff/source/wolfsrc.zip. Since then, anyone can create Wolf-like games - although now, in the era of modern 3D engines, this is no longer so interesting. :)

Why did I decide to write all this? As someone who cares about computer history, I'm a little offended that there is little documentation

on a historical game like Wolfenstein (which is at the heart of the FPS genre itself). The engine for DOOM, of course, is much more powerful - but it is also much better documented (and even in Russian language - in 1996, A. Honich's book "How to Create a Computer Game Yourself" was published, in which, de facto, DOOM technology is described). But on Wolfenstein, a cursory Internet search does not find anything (at least on the Russian). And, since at one time I dealt with the insides of this game in sufficient detail, I decided to write a little about them. I (and I think everyone who played Wolfenstein at one time) was wildly wondering: HOW did they do it? I hope that this small set of articles provides a (at least partial) answer to the question "how". Also consider it a small tribute to the brilliant developers from ID - first of all, John Romero and John Carmack - who accomplished an impressive technical feat for their time.
Corrections and criticism are accepted (and thanks are even more so :).
Yours sincerely, **trilirium**.

The main resources of the game

All external resources for Wolfenstein games are located in 8 files that have similar names (but a game-specific extension). Here are their names and contents:

- **audiohed. EXT**
  Index file to **audiot. EXT**.
- **audiot. EXT**
  All audio resources of the game, except digital samples (Digisounds).
  (In other words, all Speaker effects, FM sounds, and FM music.)
- **gamemaps. EXT**
  As you might guess from the name - cards for all levels of the game.
- **maphead. EXT**
  Index file to **gamemaps. EXT**.

- **vgadict. The EXT**
  ineach day contains 1024 bytes. Huffman dictionary for unpacking all resources from **vgagraph. EXT**.
- **vgagraph. EXT**
  All graphic assets, except textures and sprites directly from the game.
  This includes: all graphics for splash screens, menus and intermissions, status bar graphics, fonts, texts for help (Wolf3D shareware), etc.
- **vgahead. EXT**
  Index file to **vgagraph. EXT**.
- **vswap. EXT**
  Largest file – contains all the resources used in the game itself. In particular, all the basic graphics (wall textures, sprites of game objects) and Digisounds (i.e. 8-bit digital audio samples). Resources from this file take up the most space, and are loaded/unloaded dynamically as the game progresses.

Note that the graphics files are prefixed with **vga**. This (and some details of the source code) suggests that an EGA version was also planned during development! Technically, this was possible: after all, the Catacomb Abyss game was released even earlier (almost forgotten now), using Wolf-like technology, but only working in 16-color EGA mode. But in the end, they refused to support the EGA version (IMHO, it wasn't worth it).

The **EXT** extension is determined by the type of game. I am aware of the following types:

- **wl1**: Wolfenstein 3D shareware (shareware, initial episode only).
- **wl3**: Commercial wolf3d with three episodes (rare version).
- **wl6**: Complete commercial wolf3d (with six episodes).
- **sdm**: Spear of Destiny demo (free demo, only 2 levels).
- **sod**: Complete Spear of Destiny (21 levels).
- **sd2**: Spear of Destiny: Return to Danger.
- **sd3**: Spear of Destiny: Ultimate Challenge.

In fact, they differ little from the original game: a different set of maps, changed textures, sprites and voice acting for opponents (but the opponents themselves, in fact, remained the same, and in general the game engine has not changed).
Let's take a closer look at the resources. For data formats, we will use the following designation: Int16 - 16-bit integer (2 bytes), Int32 - 32-bit integer (4 bytes). The byte order in the file is Intel-endian (i.e., the low-order bytes come first).

Level Maps

Let's start with how the level maps are arranged. The information related to them is located in two files: **maphead. EXT** and **gamemaps. EXT**.
The first one is very simple: it starts with the Int16 code ABCDh (not only the signature, but also the code for RL compression of maps, see below). It is followed by a sequence of Int32 addresses that specify the order of level offsets in the **gamemaps file. EXT**. That is, the offset of the block of information about the level N in this file can be found by reading from **the maphead. EXT** is a 32-bit number at the address 2 + 4 * N. Actually, this is all that is in the **maphead**: if its size is greater than 2 + 4 * (the number of levels), then the end of the file is simply filled with zeros.
Content in **gamemaps. EXT** is more interesting. In principle, it also begins with a signature: the name and version of the program with which it was created (in most games it is the TED5 level editor). The level information block has the following structure:

- 3 addresses (Int32) for 3 "layers" of the map of this level (see below);
- 3 lengths (Int16), for the same 3 layers;
- Two numbers (Int16) that specify the width and height of the map;
- 16 bytes that specify the name for the level.

In Wolfenstein, the maps are arranged, in general, uncomplicated. Each map is a rectangular matrix of a certain width and height. In addition, for a complete description of the level, you need not one such matrix,

but two (hereinafter, we will call them "layers"). In fact (as noted above) not even two, but three layers are reserved in the file format - however, the third layer is never used (it is always empty, i.e. filled with zeros). Two layers are used - the first sets information about walls and doors, the second - about objects (static objects, objects, opponents). Each element of the layer is encoded in one byte - we will call it a "tile".
Although it is theoretically possible to write a card of any size to the file - in all games it is always 64 * 64. What's more: the original engine will simply refuse to load a map with dimensions other than these. (In general, the number 64, as we will see, is a magical :) for Wolfenstein.
The layer information itself usually precedes the level record. All layers of the map are stored in a compressed form - therefore, although in an unpacked form they always occupy 64 * 64 = 4096 bytes, but the packed size varies. Two algorithms are used sequentially to pack the layer: simple compression of repeats (RLE - Run-length encoding), followed by the use of compact coding of repeating blocks, invented by Carmack ("carmacization"). Accordingly, when unpacking, these algorithms are applied in reverse order. "Decarmacization" of the layer is performed by this function (code from the original engine):

```
#define NEARTAG 0xa7
#define FARTAG 0xa8

void CAL_CarmackExpand (unsigned far *source, unsigned far
*dest, unsigned length)
{
  unsigned ch,chhigh,count,offset;
  unsigned far *copyptr, far *inptr, far *outptr;

  length/=2;
  inptr = source;
  outptr = dest;
  while (length)
  {
    ch = *inptr++;
    chhigh = ch>>8;
    if (chhigh == NEARTAG)
    {
      count = ch&0xff;
```

```
      if (!count)
      { // have to insert a word containing the tag byte
        ch |= *((unsigned char far *)inptr)++;
        *outptr++ = ch;
        length--;
      }
      else
      {
        offset = *((unsigned char far *)inptr)++;
        copyptr = outptr - offset;
        length -= count;
        while (count--)
           *outptr++ = *copyptr++;
      }
    }
    else if (chhigh == FARTAG)
    {
      count = ch&0xff;
      if (!count)
      { // have to insert a word containing the tag byte
        ch |= *((unsigned char far *)inptr)++;
        *outptr++ = ch;
        length --;
      }
      else
      {
        offset = *inptr++;
        copyptr = dest + offset;
        length -= count;
        while (count--)
          *outptr++ = *copyptr++;
      }
    }
    else
    {
       *outptr++ = ch;
       length --;
    }
  }
}
```

The second stage - RLE-unpacking (CA_RLEWexpand) - in the original engine was generally written in assembler. Here's the equivalent C-code:

```
void CA_RLEWexpand(word *source, word *dest, long length,
word rlewtag)
{
   word value, count;
   word *end = dest + length / 2;

   /* expand it */
   do {
      if ((value = *source++) != rlewtag)
         /* uncompressed */
         *dest++ = value;
      else {
        /* compressed string */
        count = *source++;
        value = *source++;
        while (count--) *dest++ = value;
      }
   } while (dest < end);
} // CA_RLEWexpand
```

Now let's take a closer look at the information contained in the map layers.
The first layer is information about the static structure of the map (walls, doors):

- From 0 to 63: Solid Walls. The number determines which textures cover the walls of this block (strictly speaking, the number specifies two consecutive textures: for the north/south and for the east/west wall of the block;
- 22: A special case is an elevator panel. Its "activation" leads to the completion of the level. The panel only works correctly if it is in the west or east wall (so all elevators in the game are oriented "horizontally" on the map);
- From 90 to 101: different doors. Even codes specify "vertical" doors (from west to east), odd - "horizontal" (from north to south).
- 106 and above: Unallocated space (movable).

The second layer describes all the objects in the level (including the starting position of you and your opponents).

- 19..22: the player's initial position on the level (see below);
- 23..74: all static objects and objects (in brackets - the sprite number of the object):
  - 23(0): puddle of water
  - 24(1): # Green Barrel
  - 25(2): # table and chairs
  - 26(3): # green floor lamp
  - 27(4): yellow chandelier
  - 28(5): # Hanging Skeleton
  - 29(6): + bowl of dog food
  - 30(7): # white pillar
  - 31(8): # Wood in a tub
  - 32(9): reclining skeleton
  - 33(10): # washbasin (Wolf) / pole with skulls (SOD)
  - 34(11): # plant in a blue pot
  - 35(12): # Blue Urn
  - 36(13): # empty round table
  - 37(14): green pendant lamp
  - 38(15): ? kitchen utensils (Wolf) / bloody cage (SOD)
  - 39(16): # Knight's Armor
  - 40(17): # empty cell
  - 41(18): # cage with skeleton
  - 42(19): skeleton on the floor
  - 43(20): + golden key
  - 44(21): + silver key
  - 45(22): # bed (Wolf) / cage with skulls (SOD)
  - 46(23): empty basket
  - 47(24): + dish with food
  - 48(25): + first-aid kit
  - 49(26): + cartridges
  - 50 (27): + automatic
  - 51 (28): + Gatling machine gun
  - 52(29): + treasure: cross
  - 53(30): + Treasure: Cup

    - 54(31): + treasure: The Bible
    - 55(32): + Treasure: The Crown
    - 56 (33): + treasure: Megabonus (100% life + ammo)
    - 57(34): Bloody remains
    - 58(35): # wooden barrel
    - 59(36): # Full Well
    - 60(37): # empty well
    - 61(38): Pool of Blood
    - 62(39): # standing flag
    - 63(40): ? bone remains-1 (Wolf) / red pendant lamp (SOD)
    - 64(41): bone remains-2
    - 65(42): bone remains-3
    - 66(43): bone remains-4
    - 67(44): ? pots and pans (Wolf) / head on a pole (SOD)
    - 68(45): # Metal Stove (Wolf) / Blood Well (SOD)
    - 69(46): # Spear Fence (Wolf) / Hell Angel Statue (SOD)
    - 70(47): Creepers
    - 71(48): # Red Column (SOD)
    - 72(49): + cartridge box (SOD)
    - 73(50): # Truck (SOD)
    - 74(51): + Spear of Destiny! (SOD)
- 98: marker of the "secret" movable wall (pushwall);
- 99: end of level;
  (then there are the usual opponents):
- 108..111(E), 144..147(M), 180..183(H): soldier (guard), stationary;
- 112..115(E), 148..151(M), 184..187(H): guard patrolling;
- 116..119(E), 152..155(M), 188..191(H): officer, stationary;
- 120..123(E), 156..159(M), 192..195(H): officer patrolling;
- 124: Corpse of a soldier;
- 126..129(E), 162..165(M), 198..201(H): SS (SS), stationary;
- 130..133(E), 166..169(M), 202..205(H): SS (SS) patrolling;
- 134..137(E), 170..173(M), 206..209(H): dog, motionless;
- 138..141(E), 174..177(M), 210..213(H): dog, patrolling;
- 216..219(E), 234..237(M), 252..255(H): mutant, immobile;
- 220..223(E), 238..241(M), 256..259(H): mutant patrolling;
- (next are the level bosses):

- 214: Hans Grosse (Wolf, end of first episode)
- 196: Maniac Doctor Schabbs (Wolf, end of episode two)
- 224, 225, 226, 227: "Pacman Monsters" (Wolf, Episode 3, Secret Level)
- 160: Pseudo-Hitler (Wolf, end of episode three)
- 178: The Real Hitler (Wolf, end of episode three)
- 215: Otto Giftmacher (Wolf, end of episode four)
- 197: Gretel Grosse (Wolf, end of episode five)
- 179: General Fettgesicht (Wolf, end of episode six)
- 125: Trans Grosse (SOD, Level 5)
- 143: Barnacle Wilhelm (SOD, Level 10)
- 142: Ubermutant (SOD, Level 16)
- 161: Death Knight (SOD, level 18)
- 106: Ghost (SOD, Finale)
- 107: Angel of Death (SOD, finale)

Explanation of the table. For objects: the # symbol indicates that the object is blocking (the tile with the object is impassable neither for you nor for enemies). The + symbol means that the object can be taken (what happens depends on the object). Three objects marked with the symbol are blocking in Wolf, but traversable in SOD (or vice versa). There cannot be more than 400 static objects per tier.
For opponents: the letter in parentheses indicates the minimum difficulty level of the game at which this opponent appears (**E** - easy (i.e. always present), **M** - medium, **H** – hard). For the initial position of the player and for many opponents, four values are allocated - they determine the initial orientation (north, west, south, east).
A few words about the doors and the "secret" blocks of the walls. There are restrictions on the placement of doors on the map: there must be a wall to the right and left of the door (i.e. there can be no empty space or other door next to the door). Note that by "row" I mean to the west / east (when the door is horizontal) or to the north / south (when the door is vertical). That is, directly BEHIND the door, another door may well be located. In addition, there can be no more than 64 doors per level. There are really 4 types of doors: ordinary, locked with a gold key, locked with a silver key and an elevator door.

Secret walls are marked with a special marker (code 98) on the **second** layer. There is no information that sets the direction of its movement (it always moves in the direction where it was pushed). However, most maps are designed so that the player can push the secret wall in only one direction. However, there are exceptions: at the second secret level of SOD there are secret walls that can be pushed in different directions (and, by moving them incorrectly, you can cut off access to a large part of the map). Oddly enough, there are no restrictions on the number of secret walls - but this is because more than one wall cannot be in motion anyway. (If, while one of the walls is moving, you push the other, it will not move, instead you will hear the sound of NOWAYSND. You will hear it if the movement of the wall in this direction is blocked by another wall or enemy.) The movement of the wall is that it passes exactly 2 tiles and stops. However, there is a bug in the game, due to which occasionally the wall drives further (more on it later). Naturally, after the wall finishes moving, it becomes completely ordinary (and there will never be anywhere else and never :).

Finally, about code 99 ("end of level"). If the player has entered the tile marked with this code, the camera effectively turns around to face the BJ, shows how he jumps for joy, and the game ends there. This effect is only applied in Wolf, and only in the final levels of episodes 1 and 5. The rest (2, 3, 4, 6) run out immediately as soon as you kill the level boss.