Unofficial WAD3 File Spec

Table of content

Introduction

The following file specification concerns the WAD3 file format used Valve's famous GoldSrc Engine. The file extension is ".wad". Contrary to the BSP v30 file format that has been derived from the Quake 1 BSP (v29) format, the WAD3 format is tecnically fully compatible with Quake's WAD2 files. WAD files are used to store game related files in some kind of archive. Somehow they seem to contain only textures, although they are capable of holding different kinds of data.

This file spec uses constructs from the C programming language to describe the different data structures used in the WAD file format. Architecture dependent datatypes like integers are replaced by exact-width integer types of the C99 standard in the stdint.h header file, to provide more flexibillity when using x64 platforms. Basic knowledge about textures in the BSP file is recommended.

```
#include <stdint.h>
```

Header

Like almost every file also a WAD file (WAD3) starts with a specific file header which is constucted as follows:

```
typedef struct _WADHEADER
{
      char szMagic[4];      // should be WAD2/WAD3
      int32_t nDir;          // number of directory entries
      int32_t nDirOffset; // offset into directory
} WADHEADER;
```

The first 4 bytes of a WAD archive identify the file (the so-called magic number) with the 3 character string "WAD" followed by a version numer as ASCII char. Compatible formats include "WAD3" as well as "WAD2". As with the BSP file also a WAD file starts with some kind of directory containing entries similar to the lumps of the BSP file. The major difference is that a WAD file may have an arbitrary number of entries. This value is stored in the nDir member of the header followed by the offset to the array of directory entries within the file, which is usually located somewhere at the ned of the WAD file.

Directory entries

The directory of a WAD file is basically an array of structures. Every archived file has an associated entry in this directory:

```
#define MAXTEXTURENAME 16

typedef struct _WADDIRENTRY
{
      int32_t nFilePos;                  // offset in WAD
      int32_t nDiskSize;                // size in file
      int32_t nSize;                      // uncompressed size
      int8_t nType;                        // type of entry
      bool bCompression;                // 0 if none
      int16_t nDummy;                     // not used
      char szName[MAXTEXTURENAME]; // must be null terminated
} WADDIRENTRY;
```

The first value is the offset of the raw data of the archived file within the WAD file followed by the size of data used by the it in its current form. Additionally nSize stores the size of original file, if it has been compressed. This is indicated by the bCompression boolean value. If it is false, no compression is used which is mostly the case. The nType member would give as information about the type of file, that is associated with this entry. As WAD files usually contain only textures, this information may be ignored. The nDummy short integer is not used when loading textures and i have not found any information about this member. Finally the entry contains a name for the file which can be a string of maximum 16 chars including the zero termination. In case of textures, this is the name referred by the BSPMIPTEX structs of the BSP file.

Files in a WAD File - Textures

The actual files in the WAD archive, the textures, also start with a file header which may look familiar:

```
#define MAXTEXTURENAME 16
#define MIPLEVELS 4
typedef struct _BSPMIPTEX
{
      char szName[MAXTEXTURENAME];   // Name of texture
      uint32_t nWidth, nHeight;       // Extends of the texture
      uint32_t nOffsets[MIPLEVELS]; // Offsets to texture mipmaps BSPMIPTEX;
} BSPMIPTEX;
```

This struct equals the one of the BSP file exactly. The name of the texture as not needed actually, as it equals the file name in the directory entry. The next members are of particular interest. They give the size of the texture in pixel as well as the offsets into the raw texture data. The first value of the offset array points to the beginning of the original texture relativly to the local header (the BSPMIPTEX struct). From this point follows a nWidth * nHeight bytes long array of indexes (0-255) pointing to colors in a color table. The other three offsets are used the same way, but with the diference that the width and height of the image are cut in half with every further offset. These represent the so-called mipmap levels of the original image. After the last byte of the fourth mipmap there are two dummy bytes followed by the actual color table consisting of an array of triples of bytes (RGB values) with 256 elements. The indexes of the image are plugged into the color table array to receive the corresponding RGB color value for the pixel.