.tga File Extension (BIMAGE)

> The ".tga" files used in DOOM Eternal are not targa images. They are actually "BIM" (or .bimage) files.

> These files are stored in two parts. The first part consists of the `HEADER`, `MIPMAP`, and `NON_STREAMED_IMAGE` sections. This is embedded in DOOM Eternal's .resources files. The second part consists of the `STREAMED_IMAGE` section, which is embedded in Doom Eternal's .streamdb files.

Signature

DOOM Eternal ".tga" files can be identified by the first 3 bytes of the file header stored in a **.resources** file, which is always `0x42 0x49 0x4D` or `"BIM"`, short for "bimage" or "binary image". However, these files are usually stored compressed via Oodle Kraken, which may obscure the file signature.

To view these files, you can extract the .tga headers from a **.resources** file using the **EternalResourceExtractor tool**.

File Structure

A DOOM Eternal ".tga" file consists of 4 sections:

1. The `HEADER` section, which contains important details about the image size, format and encoding.
2. The `MIPMAP` section, which contains metadata for each individual image "mip"
3. The `NON_STREAMED_IMAGE` section, which contains non-streamed versions of the image. Usually, these are any "mips" that are smaller than about 32x32 pixels. Sometimes, though, it can be a full-sized image.
4. The `STREAMED_IMAGE` section, which is the portion of the image stored in .streamdb. The largest version of the image is normally stored in the .streamdb, along with several smaller mips.

> The `HEADER`, `MIPMAP`, and `NON_STREAMED_IMAGE` sections are stored as a single compressed file, embedded within a DOOM Eternal **.resources** file. The `STREAMED_IMAGE` is stored separately in a **.streamdb** file. Even though they are stored separately, they are

considered one "file" because exporting them to a usable format such as .DDS, .PNG, etc, requires that we connect all the disparate pieces.

```
struct ETERNAL_TGA_FILE
{
    HEADER header;
    MIPMAP mipmap[];□□□□□□□// Array length varies with header.MipCount;
    NON_STREAMED_IMAGE non_streamed_image[];□// Smaller mips of the image.
    STREAMED_IMAGE streamed_image[];□□□// Full-size + larger mips of the image.
};
```

Header Section

The `HEADER` struct is a 63-byte sequence:

```
struct HEADER
{□□
□char Signature[3];□□□□// "BIM"
    byte Version;□□□□□// 0x15
□int TextureType; □□□□// enum textureType_t
□int TextureMaterialKind;□□// enum textureMaterialKind_t
□int PixelWidth;□□□□□// image width in pixels
□int PixelHeight;□□□□// image height in pixels
□int Depth;
    int MipCount; □□□□□// determines # of MIPMAP structs to follow
□int64_t MipLevel;□□□□
□float unkFloat1; □□□□// usually 1.0, purpose unknown
□byte boolIsEnvironmentMap;□□// 1 if image is an environment map
□int TextureFormat;□□□□// enum textureFormat_t
□int Always7;□□□□□// literally always 7, purpose unknown
□int nullPadding;
□short AtlasPadding;
□byte boolIsStreamed;□□□// 1 if file is located in streamDB
□byte unkBool;
□byte boolNoMips;□□□□// 1 if image has no mips
□byte boolFFTBloom;□□□□// 1 if using FFT Bloom
□int StreamDBMipCount; □□□// usually same number of mips stored in streamdb
};
```

- `TextureType` is a member of the enum **`textureType_t`** - it marks the image as 2-dimensional, 3-dimensional, or cubic.
- `TextureMaterialKind` is a member of the enum **`textureMaterialKind_t`** - it tells the engine what type of material this texture is (albedo, normal, specular, etc).
- `TextureFormat` is a member of the enum **`textureFormat_t`** - it tells the engine how the image is encoded (image format, block compression type, etc).

Mipmap Section

The `MIPMAP` struct is a 36-byte sequence that comes immediately after the `HEADER`. This struct will be repeated `n` times, where `n` is equal to `HEADER.MipCount` above.

```
struct MIPMAP
{
□int64_t MipLevel;□□□□// Starts at 0, increment by 1 each time it repeats
□int MipPixelWidth;□□□□// Original PixelWidth reduced by 50% for each MipLevel
□int MipPixelHeight;□□□□// Original PixelHeight reduced by 50% for each MipLevel
□int UnknownFlagA;
□int DecompressedSize;□□□// Decompressed size in bytes
□int FlagIsCompressed;□□□// 1 if the texture is compressed
□int CompressedSize;□□□□// Compressed size in bytes
□int CumulativeSizeStreamDB;□□
};
```

- `CumulativeSizeStreamDB` is always zero for the first mipmap. For additional mipmaps, it is the sum of previous mipmaps compressed sizes.

Non-Streamed Images

The `NON_STREAMED_IMAGE` section begins immediately after the last `MIPMAP`. The **starting offset** of this section can be calculated as `offset = 63 + (36 * HEADER.MipCount)`.

Usually, any mips smaller than about 50x50 pixels in size will be stored in this section. They are packed together back-to-back, from largest mip to smallest (excluding any mips that are located in the .streamdb). The size in bytes and the pixel dimensions of each mip are given by the corresponding `MIPMAP` struct.

```
struct NON_STREAMED_IMAGE
{
```

```
□BYTE rawImageData[];□□□□// Image format and encoding given in HEADER
};
```

> Some images are "non-streaming," which means they aren't present in the .streamdb files at all. Non-streaming images will always have a `HEADER.boolIsStreamed = 0` (false). In that case, the full-sized image and all mips will be stored here, and there will not be any `STREAMED_IMAGE` section.

Streamed Images

In most cases, the full-size versions of these ".tga" images are stored in .streamdb files in a **headerless** format, where they are accessed by the game engine as needed.

```
struct STREAMED_IMAGE
{
□BYTE rawImageData[];□□□□// Image format & encoding given in HEADER
};
```

Unlike the `NON_STREAMED_IMAGE` section, these streamed images are not stored back-to-back in the **.streamdb**. Each streamed mip of the image has its own entry in the .streamdb index.

010 Editor Template

A 010 Editor template for use with Doom Eternal's TGA file headers can be found here:

https://github.com/brongo/eternal-010-templates/blob/main/templates/DoomEternalTGA.bt

Revision #25
Created 10 September 2021 12:16:24 by SamPT
Updated 13 September 2021 13:13:31 by SamPT