PS2 Half-life DOL models are closely matching PC MDL models, except that textures are stored in 8-bit PSI format and header of DOL model contains additional fields. Comparison of texture formats is shown on fig. 1.

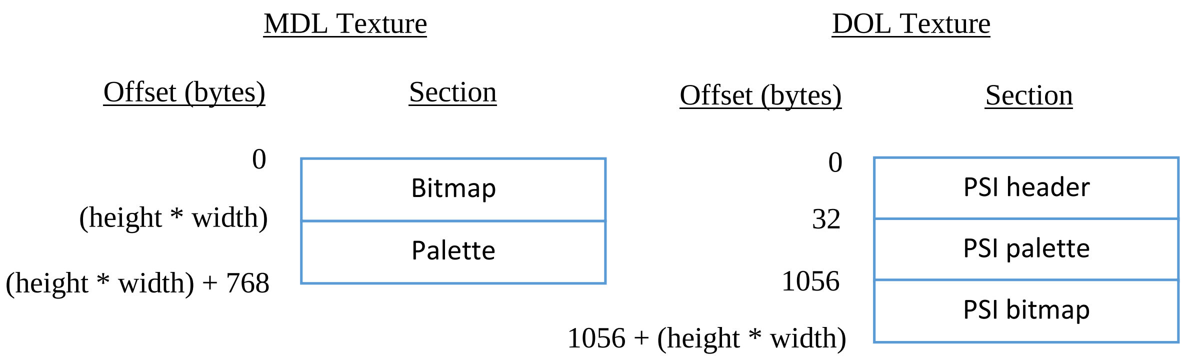

Fig. 1 – Comparison of MDL and DOL texture formats

As you can see, DOL textures are bigger than MDL textures by 288 bytes. So it is quite easy to convert DOL models to MDL format, just rewrite texture data and job is done. But because DOL textures are bigger than MDL textures, to perform conversion from MDL to DOL format you need to update:

1) Model file size.
2) Texture data offset.
3) Texture offsets in texture table.

It is important to know that textures inside DOL models must be aligned within 16-byte blocks. In addition, dimensions of DOL textures are limited to power of two values: 8, 16, 32, 64, 128, 256, etc. If input model has textures with different dimensions then they should be rescaled, otherwise either model would look corrupted or game would crash.

Additional pain of conversion from MDL to DOL format is that DOL header is bigger in size by 16 bytes. Bad header can cause improper body parts switching (for example, all scientists have Walter (Kleiner) heads or Barney shooting without pistol). If you want to resize header you must update every offset value inside model file and that is huge pain. However, I found out that it is safe to write these fields over MDL file without header enlarging, because these fields are overlapping “bone name” string that is unused in the game.

To perform more accurate conversion you can also update:

1) Internal model name (change extension)
2) Internal submodel references (change extensions)

It is also important to know that auto-aim function targets last model attachment. So if auto-aim targets wrong part of the model then consider decompiling model and adding attachment to a bone that you wish to target. But you have to keep in mind that DOL models can have **maximum of 4 attachments**, otherwise game would freeze and crash as soon as model appears on the screen.

1 MDL\DOL file structure

Simplified structure of model file is shown on fig. 2.

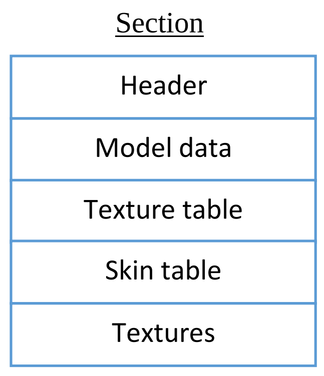

Fig. 2 – Simplified structure of model file

1.1 Header

Simplified structure of model header is listed below:

```
struct sModelHeader
{
        char Signature[4];              // "IDST"
        ulong Version;                  // 0xA - GoldSrc model
        char Name[64];                  // Internal model name
        ulong FileSize;                 // Model file size
        char SomeData1[96];             // Data that is not important for conversion
        ulong SubmodelCount;            // How many submodels
        ulong SubmodelTableOffset;      // Location of submodel table
        ulong TextureCount;             // How many textures
        ulong TextureTableOffset;       // Texture table location
        ulong TextureDataOffset;        // Textures location
        ulong SkinCount;                // How many skins
        ulong SkinEntrySize;            // Size of entry in skin table (measured in shorts)
        ulong SkinTableOffset;          // Location of skin table
        ulong SubmeshCount;             // How many submeshes (body parts)
        ulong SubmeshTableOffset;       // Location of submesh table
        char SomeData2[28];             // Data that is not important for conversion
        ulong TransitionsOffset;        // Location of transition table

        // Extra section of PS2 DOL models (missing in PC MDL models)
        ulong LODDataOffset;            // Points to a location of a LOD data section
        uchar MaxBodyParts;             // Maximum number of body parts inside one body group
        uchar NumBodyGroups;            // How many body groups are present in the model
                                        // (setting both MaxBodyParts and NumBodyGroups to 0 disables LODs)
        uchar Magic[2];                 // Filled with zeroes
        ulong FadeStart;                // Model fade start distance
        ulong FadeEnd;                  // Model fade end distance
}
```

The easiest hack to enable proper body parts switching in converted models on PS2 without resizing of the header is to set `MaxBodyParts` and `NumBodyGroups` to 0, because they are located in unused (in game) “bone name” string. I also found out that applying this hack to all “IDST” models is requirement to run game without crashes on original hardware and to stop messages like “(EE pc:002C2EDC) TLB Miss, addr=0x62c59de6 [load]” in PCSX2.

`LODDataOffset` field points to LOD data table section that contains LOD information for each body part of the model. Structure of body part entry in LOD data table looks like this:

```
struct sLODTableEntry
{
       ulong LODCount;       // Number of LODs for the specific body part
                             // (excluding full quality body part submesh (LOD0))
       ulong LODDistance1;   // Distance at which LOD1 is displayed
       ulong LODDistance2;   // Distance at which LOD2 is displayed
       ulong LODDistance3;   // Distance at which LOD3 is displayed
       ulong LODDistance4;   // Distance at which LOD4 is displayed
}
```

LOD data section contains `MaxBodyParts * NumBodyGroups` entries. Entries are grouped in body groups. If body group contains less body parts than `MaxBodyParts` then all redundant entries are present in the table but ignored. If body part has less LODs than 4 then fields for all redundant distances are present but ignored.

Distances for LODs should grow from LODDistance1 to LODDistance4. If distances are growing in opposite direction then lowest detail LOD would be always displayed no matter what. If all distances are equal then again lowest LOD would be prioritized.

LOD data section is usually located right before “transitions” section but I found out that moving LOD section to a different part of file is safe.

Let’s examine example LOD section from “agrunt.dol”. This model has 2 body groups: “body” and “arm”. First has 1 body part and the latter has 2 body parts. LOD section for this model is listed below:

LOD count	Distance 1	Distance 2	Distance 3	Distance 4	
`03 00 00 00`	`F4 01 00 00`	`E8 03 00 00`	`A4 06 00 00`	`00 00 00 00`	`<- only “body” body part`
`00 00 00 00`	`00 00 00 00`	`00 00 00 00`	`00 00 00 00`	`00 00 00 00`	`<- placeholder`
`03 00 00 00`	`F4 01 00 00`	`E8 03 00 00`	`A4 06 00 00`	`00 00 00 00`	`<- first “arm” body part`
`03 00 00 00`	`F4 01 00 00`	`E8 03 00 00`	`A4 06 00 00`	`00 00 00 00`	`<- second “arm” body part`

This section tells us that each body part has 3 LODs. Distances at which LODs are appearing: 500, 1000, 1700 units (0x1F4, 0x3E8, 0x6A4).

1.2 Texture table

Texture table must have entry for each texture. Structure of texture table entry looks like this:

```
struct sTextureTableEntry
{
	char Name[68];			// Texture name
	ulong Width;			// Texture width
	ulong Height;			// Texture height
	ulong Offset;			// Location of texture in model file
}
```

1.3 Skin table

Skin table must have entry for each skin. Structure of skin table entry is listed below:

```
struct sSkinTableEntry
{
	short TexIndex[SkinTexCount];	// Each short contains index of texture
					// that is associated with this skin
}
```

1.4 Textures

Actual texture data is located in this section. Structure of textures is shown on fig. 1.