MD2 (file format)

MD2 is a model format used by id Software's id Tech 2 engine and is thus used by Quake II as well as many other games, most of them using this engine, including SiN and Soldier of Fortune.

The format is primarily used for animated player models although it can also be used for static models.

Unlike more recent character model formats, MD2 animations are achieved via keyframes on a per-vertex level; the keyframes are stored within the model file and the engine interpolates between them to create a smooth animation.

MD2 はモデル形式の一つであり、id Software の id Tech 2 エンジンで使われ、よって Quake II を始めとするゲームらで使われた。このエンジンを使った主な例は、SiN と Soldier of Fortune だ。

この形式は第一には、アニメするプレイヤーのモデルらで使われたが、静的なモデルらのためにも使うことができる。

より最近のキャラクターモデル形式とは異なり、MD2 のアニメーションらは、頂点単位のキーフレームらで達成される。キーフレームらはモデルファイルに蓄えられており、エンジンは滑らかなアニメを生むためにそれらの間を補間する。

File format ファイル形式

An MD2 file begins with a fixed length header followed by static model data such as texture coordinates.

Dynamic data such as vertices and normals are stored within a number of file chunks called frames (or key-frames) which each have their own short headers.

一つの MD2 ファイルは固定長のヘッダから始まり、テクスチャ座標などの静的なモデルデータがそれに続く。

頂点らや法線らといった動的なデータは、フレーム（またはキーフレーム）と呼ばれるいくつかのファイルの塊のなかに格納される。その各々が自らの短いヘッダを持っている。

In defining the file structure several data types will be referred to.

int (4 bytes), short (2 bytes), and char (1 byte)

ファイル構造の定義においては次のデータ型らが言及される。

int は 4、short は 2、char は 1 バイトだ。

To recover the floating-point texture coordinates as used by common 3D display API's such as OpenGL, divide the texture coordinates by the respective size dimensions of the texture:

OpenGL などの一般的な 3D 表示 API で使われるような、浮動小数点でのテクスチャ座標を復元するためには、テクスチャのそれぞれの次元の大きさで対応するテクスチャ座標を除算する。

```
sfloat = (float)s / texturewidth
```

オフセット	データ型	名前	説明
0	int	ident	“IDP2” に等しくなければならない
4	int	version	MD2 のバージョン。8 でなければならない。
8	int	skinwidth	テクスチャの幅
12	int	skinheight	テクスチャの高さ
16	int	framesize	1 つのフレームのバイトサイズ
20	int	num_skins	テクスチャの数
24	int	num_xyz	頂点の数
28	int	num_st	テクスチャ座標の数
32	int	num_tris	三角形の数
36	int	num_glcmds	OpenGL コマンドの数
40	int	num_frames	フレームの合計個数
44	int	ofs_skins	スキン名らのオフセット（各スキン名は null 終端の unsigned char[64]）
48	int	ofs_st	s-t テクスチャ座標へのオフセット
52	int	ofs_tris	三角形らへのオフセット
56	int	ofs_frames	フレームのデータへのオフセット
60	int	ofs_glcmds	OpenGL コマンドらへのオフセット
64	int	ofs_end	ファイル終端へのオフセット

表 1　MD2 Header

データ型	名前
short	s
short	t

表 2　オフセット ofs_st の位置に num_st 個ある構造

```
tfloat = (float)t / textureheight
```

At offset ofs_tris there are num_tris of the following structure

オフセット ofs_tris の位置には num_tris 個の次の構造がある。

```
float scale[3]
float translate[3]
char name[16]
```

Then there is num_xyz of this structure:

そして num_xyz 個、次の構造がある。

```
unsigned char v[3]
unsigned char lightnormalindex
```

Each vertex is stored as an integer array.
To recover the floating-point vertex coordinates,

各頂点は整数の配列に格納されている。
浮動小数点な頂点座標を復元するためには、MD2

the MD2 reader multiplies each coordinate by the scaling vector for the current frame and then adds the frame's translation vector; these vectors can be found in the frame's header.

Alternatively, you can set the translation/scale matrix with model's values before rendering the model's frame (e.g. if using OpenGL); please note that scaling using glScale may de-normalize the surface normals.

を読み込んだ者は、現在のフレームのスケーリングベクトルで各座標を乗算し、そのフレームのトランスレーションベクトルを追加する。それらのベクトルはフレームのヘッダにあるかもしれない。

あるいは、モデルの値を用いてトランスレーション／スケーリング行列を、（例えば OpenGL を用いるなら）モデルのフレームに設定することもできる。なお、glScale を用いたスケーリングは面法線を非正規化することがあることに注意せよ。

Example 例

This is an example of how to read a single frame and display it, written in pseudocode.

次に、一つのフレームを読んで表示する方法を、疑似コードで例示する。

```
get struct triangle[], int num_tris, unsigned char vertex[],
    float scale[3], float translate[3]

int index=0
int j=0
short tri

loop while index < num_tris
  loop while j < 3

    tri = triangle[ index ].textureindex[j]

    texture_function_s texture_coordinates[ tri ].s / skinwidth
    texture_function_t texture_coordinates[ tri ].t / skinheight

    tri = triangle[ index ].vertexindex[j]

    normal_function vertex[ tri ].lightnormalindex

    vertex_function_x (vertex[ tri ].v[0] * scale[0]) + translate[0]
    vertex_function_y (vertex[ tri ].v[1] * scale[1]) + translate[1]
    vertex_function_z (vertex[ tri ].v[2] * scale[2]) + translate[2]

    j = j + 1
  end loop
```

```
  index = index + 1
end loop
```

examination

.md2 ファイルを実際に走査して offset を増加しつつ妥当な値が得られることを次のコードで確認した。

Python code

```
import array
import collections
import struct
path = "sydney.md2"
with open(path, mode="rb") as f:
    bytes=f.read()
values = struct.unpack_from(17*"I", bytes, 0)
ofs = 17*4
keys = ("ident version skinwidth skinheight framesize num_skins num_xyz "
        "num_st num_tris num_glcmds num_frames ofs_skins ofs_st ofs_tris "
        "ofs_frames ofs_glcmds ofs_end").split()
d = collections.OrderedDict(zip(keys, values))
desc = {}
desc["ident"]      = ’識別子。必ず"IDP2"(844121161)。’
desc["version"]    = "必ず 8"
desc["skinwidth"]  = "テクスチャの幅"
desc["skinheight"] = "テクスチャの高さ"
desc["framesize"]  = "1 つのフレームのバイトサイズ"
desc["num_skins"]  = "テクスチャの数"
desc["num_xyz"]    = "頂点の数"
desc["num_st"]     = "テクスチャ座標の数"
desc["num_tris"]   = "三角形の数"
desc["num_glcmds"] = "OpenGL コマンドの数"
desc["num_frames"] = "フレームの合計個数"
desc["ofs_skins"]  = "スキン名らのオフセット"
desc["ofs_st"]     = "s-t テクスチャ座標へのオフセット"
desc["ofs_tris"]   = "三角形らへのオフセット"
desc["ofs_frames"] = "フレームのデータへのオフセット"
desc["ofs_glcmds"] = "OpenGL コマンドらへのオフセット"
desc["ofs_end"]    = "ファイル終端へのオフセット"
```

```
for k, v in d.items():
    s = "  " + desc[k] if k in desc else ""
    print("{:<10}{:>15}{}".format(k, v, s))

ss = []
ts = []
for i in range(d["num_st"]):  # テクスチャ座標ら
    s, t = struct.unpack_from("hh", bytes, ofs)
    ss.append(s)
    ts.append(t)
    # print("st{}_{}_{} ".format(i, s, t), end="")
    ofs += 2*2
print("ofs=", ofs)
def f():
    """テクスチャ座標らを散布図としてプロットする。"""
    import matplotlib.pyplot as plt
    plt.scatter(ss, ts)
    plt.show()
# f()

for i in range(d["num_tris"]):  # 三角形ら
    xs = struct.unpack_from("3h3h", bytes, ofs)
    vertexindex = xs[0:3]   # 3 つの頂点それぞれの頂点 ID
    textureindex = xs[3:6]  # 3 つの頂点それぞれのテクスチャ座標 ID
    # print(vertexindex, textureindex)
    ofs += 3*2 + 3*2
print("ofs=", ofs)

for i in range(d["num_frames"]):  # キーフレームら
    xs = struct.unpack_from("3f3f16c", bytes, ofs)
    scale = xs[0:3]
    translate = xs[3:6]
    name = b"".join(xs[6:]).decode("utf-8").strip("\0")
    # print(scale, translate, name)
    ofs += 3*4 + 3*4 + 16*1
    for j in range(d["num_xyz"]):
        xs = struct.unpack_from("3BB", bytes, ofs)
        v = xs[0:3]
        lightnormalindex = xs[3]
        # print(v, lightnormalindex)
```

```
        ofs += 3*1 + 1
print("ofs=", ofs)

i_primitive = 0
# for i in range(d["num_glcmds"]):
while True:  # OpenGL コマンドら
    n_vertices, = struct.unpack_from("i", bytes, ofs)
    ofs += 4
    if n_vertices == 0:
        break
    if n_vertices > 0:
        primitive = "GL_TRIANGLE_STRIP"
    else:
        primitive = "GL_TRIANGLE_FAN"
    n_vertices = abs(n_vertices)
    # print(i_primitive, n_vertices, primitive)
    for j in range(n_vertices):
        s, t, index = struct.unpack_from("ffI", bytes, ofs)
        # print("   ", s, t, index)
        ofs += 3*4
    i_primitive += 1
print("ofs=", ofs)
```