

Spécifications du format MDL (modèles Quake)

par [David Henry](#)

Date de publication : 20/12/2004

Dernière mise à jour : 19/12/2005

Cet article a pour objectif d'expliquer comment charger les modèles MDL de
Quake. Les exemples donnés dans ce document sont écrits en C.

- 1 - Introduction
- 2 - L'en-tête
- 3 - Types de données
 - 3.1 - Vecteur
 - 3.2 - Informations de texture
 - 3.3 - Coordonnées de texture
 - 3.4 - Triangles
 - 3.5 - Sommets
 - 3.6 - Frames
- 4 - Lecture d'un fichier MDL
- 5 - Rendu du modèle
- 6 - Animation
- 7 - Constantes

1 - Introduction

Le format MDL est le format de modèle utilisé dans Quake (juin 1996). Un fichier modèle MDL présente les caractéristiques suivantes :

- Données géométriques du modèle (triangles) ;
- Données de texture 8 bits ;
- Animations par frame.

Un fichier MDL peut contenir plusieurs textures.

L'extension de fichier des modèles MDL est « mdl ». Un fichier MDI est un fichier binaire composé de deux parties : l'en-tête et les données. L'en-tête du fichier apporte des informations sur les données afin de pouvoir les manipuler.

En-tête
Données

Les types de variables utilisés ici ont les tailles suivantes :

- **char** : 1 octet
- **short** : 2 octets
- **int** : 4 octets
- **float** : 4 octets
- **ubyte** : 1 octet non signé

2 - L'en-tête

L'en-tête (header en anglais) est contenu dans une structure située au début du fichier :

```

/* mdl header */
typedef struct
{
    int    ident;           /* magic number: "IDPO" */
    int    version;        /* version: 6 */

    vec3_t scale;          /* scale factor */
    vec3_t translate;      /* translation vector */
    float  boundingradius; /* eyes' position */
    vec3_t eyeeposition;

    int    num_skins;      /* number of textures */
    int    skinwidth;      /* texture width */
    int    skinheight;     /* texture height */

    int    num_verts;      /* number of vertices */
    int    num_tris;       /* number of triangles */
    int    num_frames;     /* number of frames */

    int    synctype;       /* 0 = synchron, 1 = random */
    int    flags;          /* state flag */
    float  size;

} mdl_header_t;

```

ident est le numéro magique du fichier. Il sert à identifier le type de fichier. *ident* doit être égal à 1330660425 ou à "IDPO". On peut obtenir la valeur numérique avec l'expression $((('O' << 24) + ('P' << 16) + ('D' << 8) + 'I')$.

version est le numéro de version. Il doit être égal à 6.

scale et *translate* servent à obtenir les coordonnées réelles des sommets du modèle. *scale* est le facteur de redimensionnement et *translate* le vecteur de translation (ou l'origine du modèle). Il faut d'abord multiplier chaque composantes des coordonnées du sommet par la composante respective de *scale*, puis ajouter la composante respective de *translate* :

```
vreel[i] = (scale[i] * vertex[i]) + translate[i];
```

où *i* varie de 0 à 2 (composantes x, y et z).

boundingradius est le rayon d'une sphère dans laquelle le modèle tout entier peut-être contenu (utilisé pour la détection de collision par exemple).

eyeeposition est la position des yeux (s'il s'agit d'un monstre ou personnage). Vous en faites ce que vous voulez.

num_skins est le nombre de textures présentent dans le fichier. *skinwidth* et *skinheight* sont respectivement la largeur et la hauteur de la texture du modèle. Toutes les textures doivent avoir les mêmes dimensions.

num_verts est le nombre de sommets d'une frame du modèle.

num_tris est le nombre de triangles du modèle.

num_frames est le nombre de frames que possède le modèle.

3 - Types de données

3.1 - Vecteur

Le vecteur, composé de trois coordonnées flottantes (x, y, z) :

```
/* vector */
typedef float vec3_t[3];
```

3.2 - Informations de texture

Les informations de texture viennent directement après l'en-tête du modèle dans le fichier. Il peut s'agir d'une texture composée d'une seule images ou d'un groupe d'images (texture animée).

```
/* skin */
typedef struct
{
    int      group;    /* 0 = single */
    GLubyte *data;    /* texture data */
} mdl_skin_t;
```

ou :

```
/* group of pictures */
typedef struct
{
    int      group;    /* 1 = group */
    int      nb;       /* number of pics */
    float    *time;    /* time duration for each pic */
    GLubyte **data;    /* texture data */
} mdl_groupskin_t;
```

time est un tableau de dimension *nb* et *data* est un tableau de *nb* tableaux de dimensions *skinwidth* * *skinheight*.

Les données des images sont contenues dans le tableau *data* et sont des images en mode index couleur sur 8 bits. La palette de couleur se trouve généralement dans un fichier LMP (*.Imp). Les fichiers LMP sont des fichiers binaires contenant la palette sur 768 octets (256 couleurs sur 24 bits). Ils ne contiennent rien d'autre.

Une palette de couleur est [disponible](#) sous format texte.

Il y a *num_skins* objets de type *mdl_skin_t* ou *mdl_groupskin_t*.

3.3 - Coordonnées de texture

Les coordonnées de texture sont regroupées dans une structure et sont stockées sous forme de *short* :

```
/* texture coords */
typedef struct
{
    int onseam;
    int s;
    int t;
} mdl_texCoord_t;
```

Les textures sont généralement découpées en deux parties : l'une pour le devant du modèle et l'autre pour le dos. La partie dorsale doit être décalée de $skinwidth/2$ par rapport à la partie frontale.

onseam indique si le sommet est situé sur la frontière entre la partie frontale et la partie dorsale du modèle (un peu comme un trait de couture).

Pour obtenir les coordonnées (s, t) réelles (sur un intervalle de 0,0 à 1,0), il faut leur ajouter 0,5 et diviser le résultat par *skinwidth* pour s et *skinheight* pour t.

Il y a *num_verts* couples (s, t) de coordonnées de texture dans un modèle MDL. Ces données viennent après les données de texture.

3.4 - Triangles

Les triangles possèdent chacun un tableau d'indices de sommets et un drapeau permettant de savoir s'il est situé sur la face avant ou sur la face arrière du modèle.

```
/* triangle info */
typedef struct
{
    int facesfront; /* 0 = backface, 1 = frontface */
    int vertex[3]; /* vertex indices */
} mdl_triangle_t;
```

Dans le cas où un sommet est situé sur la couture entre les deux parties et faisant partie d'un triangle de la face arrière, il faut ajouter $skinwidth/2$ à s pour corriger les coordonnées de texture.

Il y a *num_tris* triangles dans un modèle MDL. Les données des triangles suivent les données de coordonnées de texture dans le fichier.

3.5 - Sommets

Les sommets sont composés d'un triplet de coordonnées « compressées » stockés sur un octet par composante, et d'un index de vecteur normal. Le tableau de normales se trouve dans le fichier [anorms.h](#) de Quake et est composé de 162 vecteurs en coordonnées flottantes (3 *float*).

```
/* compressed vertex */
typedef struct
{
    unsigned char v[3];
    unsigned char normalIndex;
} mdl_vertex_t;
```

3.6 - Frames

Les frames possèdent une liste de sommets et quelques autres informations spécifiques.

```
/* simple frame */
typedef struct
{
    mdl_vertex_t bboxmin; /* bounding box min */
    mdl_vertex_t bboxmax; /* bounding box max */
    char        name[16];
    mdl_vertex_t *verts; /* vertex list of the frame */
} mdl_simpleframe_t;
```

bboxmin et *bboxmax* définissent un volume dans lequel le modèle peut être entièrement contenu. *name* est le nom de la frame. *verts* est la liste des sommets de la frame.

Les frames peuvent être des frames simples ou des groupes de frames. Elles sont identifiées par une variable *type* qui vaut 0 pour une frame simple, et une valeur non nulle pour un groupe de frames :

```
typedef struct
{
    int          type; /* 0 = simple */
    mdl_simpleframe_t frame;
} mdl_frame_t;
```

ou :

```
typedef struct
{
    int          type; /* !0 = group */
    mdl_vertex_t min; /* min pos in all simple frames */
    mdl_vertex_t max; /* max pos in all simple frames */
    float        *time; /* time duration for each frame */
    mdl_simpleframe_t *frames; /* simple frame list */
} mdl_groupframe_t;
```

time et *frames* sont de dimension *nb*. *min* et *max* correspondent aux positions minimum et maximum parmi tous les sommets de toutes les frames. *time* est la durée de chaque frame.

Il y a *num_frames* frames dans un modèle MDL. Les données des frames suivent les données des triangles dans un fichier MDL.

4 - Lecture d'un fichier MDL

En supposant que *mdl_model_t* est une structure contenant les données d'un modèle MDL, et que **mdl* est un pointeur sur une zone mémoire déjà allouée, voici un exemple de fonction lisant les données d'un fichier MDL :

```
int
ReadMDLModel (const char *filename, mdl_model_t *mdl)
{
    FILE *fp;
    int i;

    fp = fopen (filename, "rb");
    if (!fp)
    {
        fprintf (stderr, "error: couldn't open \"%s!\", filename);
        return 0;
    }

    /* read header */
    fread (&mdl->header, 1, sizeof (mdl_header_t), fp);

    if ((mdl->header.ident != 1330660425) ||
        (mdl->header.version != 6))
    {
        /* error! */
        fclose (fp);
        return 0;
    }

    /* memory allocation */
    mdl->skins = (mdl_skin_t *)malloc (sizeof (mdl_skin_t) * mdl->header.num_skins);
    mdl->texcoords = (mdl_texCoord_t *)malloc (sizeof (mdl_texCoord_t) *
mdl->header.num_verts);
    mdl->triangles = (mdl_triangle_t *)malloc (sizeof (mdl_triangle_t) *
mdl->header.num_tris);
    mdl->frames = (mdl_frame_t *)malloc (sizeof (mdl_frame_t) * mdl->header.num_frames);
    mdl->tex_id = (GLuint *)malloc (sizeof (GLuint) * mdl->header.num_skins);

    mdl->iskin = 0;

    /* read texture data */
    for (i = 0; i < mdl->header.num_skins; ++i)
    {
        mdl->skins[i].data = (GLubyte *)malloc (sizeof (GLubyte)
            * mdl->header.skinwidth * mdl->header.skinheight);

        fread (&mdl->skins[i].group, sizeof (int), 1, fp);
        fread (mdl->skins[i].data, sizeof (GLubyte),
            mdl->header.skinwidth * mdl->header.skinheight, fp);

        mdl->tex_id[i] = MakeTexture (i, mdl);

        free (mdl->skins[i].data);
        mdl->skins[i].data = NULL;
    }

    fread (mdl->texcoords, sizeof (mdl_texCoord_t), mdl->header.num_verts, fp);
    fread (mdl->triangles, sizeof (mdl_triangle_t), mdl->header.num_tris, fp);

    /* read frames */
    for (i = 0; i < mdl->header.num_frames; ++i)
    {
        /* memory allocation for vertices of this frame */
        mdl->frames[i].frame.verts = (mdl_vertex_t *)
            malloc (sizeof (mdl_vertex_t) * mdl->header.num_verts);

        /* read frame data */
        fread (&mdl->frames[i].type, sizeof (long), 1, fp);
        fread (&mdl->frames[i].frame.bboxmin, sizeof (mdl_vertex_t), 1, fp);
    }
}
```

```
        fread (&mdl->frames[i].frame.bboxmax, sizeof (mdl_vertex_t), 1, fp);
        fread (mdl->frames[i].frame.name, sizeof (char), 16, fp);
        fread (mdl->frames[i].frame.verts, sizeof (mdl_vertex_t),
                mdl->header.num_verts, fp);
    }
    fclose (fp);
    return 1;
}
```

Remarque : ce code ne peut gérer les fichiers MDL composé de groupes de frames.

5 - Rendu du modèle

Exemple de code pour le rendu d'une frame n d'un modèle mdl :

```

void
RenderFrame (int n, mdl_model_t *mdl)
{
    int i, j;
    GLfloat s, t;
    vec3_t v;
    mdl_vertex_t *pvert;

    /* check if n is in a valid range */
    if ((n < 0) || (n > mdl->header.num_frames - 1))
        return;

    /* enable model's texture */
    glBindTexture (GL_TEXTURE_2D, mdl->tex_id[mdl->iskin]);

    /* draw the model */
    glBegin (GL_TRIANGLES);
    /* draw each triangle */
    for (i = 0; i < mdl->header.num_tris; ++i)
    {
        /* draw each vertex */
        for (j = 0; j < 3; ++j)
        {
            pvert = &mdl->frames[n].frame.verts[mdl->triangles[i].vertex[j]];

            /* compute texture coordinates */
            s = (GLfloat)mdl->texcoords[mdl->triangles[i].vertex[j]].s;
            t = (GLfloat)mdl->texcoords[mdl->triangles[i].vertex[j]].t;

            if (!mdl->triangles[i].facesfront &&
                mdl->texcoords[mdl->triangles[i].vertex[j]].onseam)
            {
                s += mdl->header.skinwidth * 0.5f; /* backface */
            }

            /* scale s and t to range from 0.0 to 1.0 */
            s = (s + 0.5) / mdl->header.skinwidth;
            t = (t + 0.5) / mdl->header.skinheight;

            /* pass texture coordinates to OpenGL */
            glTexCoord2f (s, t);

            /* normal vector */
            glNormal3fv (anorms_table [pvert->normalIndex]);

            /* calculate real vertex position */
            v[0] = (mdl->header.scale[0] * pvert->v[0]) + mdl->header.translate[0];
            v[1] = (mdl->header.scale[1] * pvert->v[1]) + mdl->header.translate[1];
            v[2] = (mdl->header.scale[2] * pvert->v[2]) + mdl->header.translate[2];

            glVertex3fv (v);
        }
    }
    glEnd ();
}

```

6 - Animation

L'animation du modèle se fait par frame. Une frame est une séquence d'une animation. Pour éviter les saccades, on procède à une interpolation linéaire entre les coordonnées du sommet de la frame actuelle et celles de la frame suivante (de même pour le vecteur normal) :

```
mdl_vertex_t *pvert1, *pvert2;
vec3_t v;

for (/* ... */)
{
    pvert1 = &mdl->frames[ actuel ].frame.verts[ mdl->triangles[i].vertex[j] ];
    pvert2 = &mdl->frames[ actuel + 1 ].frame.verts[ mdl->triangles[i].vertex[j] ];

    /* ... */

    v[0] = mdl->header.scale[0] * (pvert1->v[0] + interp * (pvert2->v[0] -
pvert1->v[0])) + mdl->header.translate[0];
    v[1] = mdl->header.scale[1] * (pvert1->v[1] + interp * (pvert2->v[1] -
pvert1->v[1])) + mdl->header.translate[1];
    v[2] = mdl->header.scale[2] * (pvert1->v[2] + interp * (pvert2->v[2] -
pvert1->v[2])) + mdl->header.translate[2];

    /* ... */
}
```

v est le sommet final à dessiner. *interp* est le pourcentage d'interpolation entre les deux frames. C'est un *float* compris entre 0,0 et 1,0. Lorsqu'il vaut 1,0, *actuel* est incrémenté de 1 et *interp* est réinitialisé à 0,0. Il est inutile d'interpoler les coordonnées de texture, car ce sont les même pour les deux frames. Il est préférable que *interp* soit fonction du nombre d'images par seconde sorti par le programme.

```
void
Animate (int start, int end, int *frame, float *interp)
{
    if ((*frame < start) || (*frame > end))
        *frame = start;

    if (*interp >= 1.0f)
    {
        /* move to next frame */
        *interp = 0.0f;
        (*frame)++;

        if (*frame >= end)
            *frame = start;
    }
}
```

7 - Constantes

Quelques constantes définissant des dimensions maximales :

- Nombre maximum de triangles : 2048
- Nombre maximum de sommets : 1024
- Nombre maximum de coordonnées de texture : 1024
- Nombre maximum de frames : 256
- Nombre de normales précalculées : 162

Code source d'exemple : [mdl.c](#) (13,5 Ko), [anorms.h](#) (6,7 Ko), [colormap.h](#) (4,3 Ko)

Version PDF : [télécharger](#) (40.3 Ko)