3D GameStudio
A4 Programmer's Manual

Contents

The A5 DDL interface

DLLs are customer-programmed extensions to the engine and to the WDL language. They can be used with the commercial and professional editions, but can only be created with the SDK (source development kit) that comes with the professional edition. This way, professional edition owners can create arbitrary DLLs for adding new effects, actor AI or WDL instructions, and distribute or sell them to other 3D GameStudio users.

The Microsoft Visual C++ development system is required for creating DLL extensions. The DLL SDK contains an interface library that must be linked to any DLL. An example VC++ project with a DLL template is also provided, which makes it pretty easy to create extensions even for not-so-experienced C programmers who have never used DLLs before.

DLL extensions work bidirectionally: WDL instructions can access DLL functions, and DLL functions can access essential engine functions and variables. On opening a DLL, the engine transfers the pointer to an internal interface structure to the interface library. The interface contains pointers to engine variables and functions, like the frame buffer, the DirectX interface, the network interface, the DirectInput interface, the level, the WDL functions and so on. Theoretically everything - MP3 or MOD players, a physics engine, another 3D engine or even another scripting language - could be added to the engine this way.

On accessing system resources like sound, video, joystick and so on, the DLL must take care of possible resource conflicts. The engine shares its resources and expects the same from the code inside the DLL. For instance, code that requires exclusive access to the sound device (like some old MOD players) won't work. Some resources (like the midi player) can't be shared - if midi music is played by the DLL, the engine must not play a midi file at the same time and vice versa. Also care must be taken that for writing something into the frame buffer it must be locked before, and unlocked afterwards. The interface library provides functions for that.

WDL object structures

Pointers to WDL objects can be transferred to DLL functions, thus allowing object manipulation. The internal engine format of important WDL objects is listed here.

```
typedef long fixed;   // internal 22.10 fixed point number format

typedef struct {
   long  index;   // A4 internal use only
   char  *name;   // A4 internal use only
   long  next;    // A4 internal use only
   long  modelindex; // A4 internal use only

// the following variables can be externally used by WDL or DLL functions
   fixed x,y,z;         // position of the entity
   fixed pan,tilt,roll; // angles
   fixed scale_x,scale_y,scale_z;
   long  flags;
   fixed frame;         // model or sprite frame
   fixed nextframe;     // next frame for inbetweening
   fixed skin;          // skin number
   fixed ambient;       // brightness added to the entity, in percent
   fixed alpha;         // transparency of the entity, in percent
   fixed lightrange;    // range of dynamic light emitted by the entity
   fixed red,green,blue; // colour of dynamic light emitted by the entity
   long  emask;         // event enable flags
   long  eflags;        // internal status flags
```

```
   fixed min[3],max[3];  // bounding box
   fixed trigger_range;
   fixed push;           // collision behaviour
   fixed shadow_range;   // range within a shadow is projected onto to the floor
   fixed floor_range;    // range within the entity reflects light from the floor
   long  client_id;   // client who has created this entity (when in multiplayer mode)
   fixed skill[48];   // entity skills
} A4_ENTITY;
```

(to be continued)

The WMP map format

WMP files are plain ASCII files that can be viewed with any text editor. They contain blocks, entities, and objects that are built of them or of further objects. This is a typical WMB file:

```
// WMP2
// ACKWED V 4.39
// Created 11.09.2000

blocks 1
entities 2
objects 4

wed {
    palette {
        00 00 00  0f 0f 0f  1f 1f 1f  2f 2f 2f  3f 3f 3f  4b 4b 4b  5b 5b 5b  6b 6b 6b
        7b 7b 7b  8b 8b 8b  9b 9b 9b  ab ab ab  bb bb bb  cb cb cb  db db db  eb eb eb
        0f 0b 07  17 0f 0b  1f 17 0b  27 1b 0f  2f 23 13  37 2b 17  3f 2f 17  4b 37 1b
        53 3b 1b  5b 43 1f  63 4b 1f  6b 53 1f  73 57 1f  7b 5f 23  83 67 23  8f 6f 23
        0b 0b 0f  13 13 1b  1b 1b 27  27 27 33  2f 2f 3f  37 37 4b  3f 3f 57  47 47 67
        4f 4f 73  5b 5b 7f  63 63 8b  6b 6b 97  73 73 a3  7b 7b af  83 83 bb  8b 8b cb
        00 00 00  07 07 00  0b 0b 00  13 13 00  1b 1b 00  23 23 00  2b 2b 07  2f 2f 07
        37 37 07  3f 3f 07  47 47 07  4b 4b 0b  53 53 0b  5b 5b 0b  63 63 0b  6b 6b 0f
        07 00 00  0f 00 00  17 00 00  1f 00 00  27 00 00  2f 00 00  37 00 00  3f 00 00
        47 00 00  4f 00 00  57 00 00  5f 00 00  67 00 00  6f 00 00  77 00 00  7f 00 00
        13 13 00  1b 1b 00  23 23 00  2f 2b 00  37 2f 00  43 37 00  4b 3b 07  57 43 07
        5f 47 07  6b 4b 0b  77 53 0f  83 57 13  8b 5b 13  97 5f 1b  a3 63 1f  af 67 23
        23 13 07  2f 17 0b  3b 1f 0f  4b 23 13  57 2b 17  63 2f 1f  73 37 23  7f 3b 2b
        8f 43 33  9f 4f 33  af 63 2f  bf 77 2f  cf 8f 2b  df ab 27  ef cb 1f  ff f3 1b
        0b 07 00  1b 13 00  2b 23 0f  37 2b 13  47 33 1b  53 37 23  63 3f 2b  6f 47 33
        7f 53 3f  8b 5f 47  9b 6b 53  a7 7b 5f  b7 87 6b  c3 93 7b  d3 a3 8b  e3 b3 97
        ab 8b a3  9f 7f 97  93 73 87  8b 67 7b  7f 5b 6f  77 53 63  6b 4b 57  5f 3f 4b
        57 37 43  4b 2f 37  43 27 2f  37 1f 23  2b 17 1b  23 13 13  17 0b 0b  0f 07 07
        bb 73 9f  af 6b 8f  a3 5f 83  97 57 77  8b 4f 6b  7f 4b 5f  73 43 53  6b 3b 4b
        5f 33 3f  53 2b 37  47 23 2b  3b 1f 23  2f 17 1b  23 13 13  17 0b 0b  0f 07 07
        db c3 bb  cb b3 a7  bf a3 9b  af 97 8b  a3 87 7b  97 7b 6f  87 6f 5f  7b 63 53
        6b 57 47  5f 4b 3b  53 3f 33  43 33 27  37 2b 1f  27 1f 17  1b 13 0f  0f 0b 07
        6f 83 7b  67 7b 6f  5f 73 67  57 6b 5f  4f 63 57  47 5b 4f  3f 53 47  37 4b 3f
        2f 43 37  2b 3b 2f  23 33 27  1f 2b 1f  17 23 17  0f 1b 13  0b 13 0b  07 0b 07
        ff f3 1b  ef df 17  db cb 13  cb b7 0f  bb a7 0f  ab 97 0b  9b 83 07  8b 73 07
        7b 63 07  6b 53 00  5b 47 00  4b 37 00  3b 2b 00  2b 1f 00  1b 0f 00  0b 07 00
        00 00 ff  0b 0b ef  13 13 df  1b 1b cf  23 23 bf  2b 2b af  2f 2f 9f  2f 2f 8f
        2f 2f 7f  2f 2f 6f  2f 2f 5f  2b 2b 4f  23 23 3f  1b 1b 2f  13 13 1f  0b 0b 0f
        2b 00 00  3b 00 00  4b 07 00  5f 07 00  6f 0f 00  7f 17 07  93 1f 07  a3 27 0b
        b7 33 0f  c3 4b 1b  cf 63 2b  db 7f 3b  e3 97 4f  e7 ab 5f  ef bf 77  f7 d3 8b
        a7 7b 3b  b7 9b 37  c7 c3 37  e7 e3 57  7f bf ff  ab e7 ff  d7 ff ff  67 00 00
        8b 00 00  b3 00 00  d7 00 00  ff 00 00  ff f3 93  ff f7 c7  ff ff ff  9f 5b 53
    }

    wad {
        0 0 0 0 0
        <standard.wad> 1
    }

    bookmark {
        *waterblue
        earthtile
        ;
        ;
        ;
        ;
        ;
    }

    view {
        xy.1 -1 52 379 327 0.0 0.0 0.0 1536.0 0.745
        3d.1 379 52 763 327 250.0 -250.0 250.0 1536.0 -0.785398 -0.61548 0.0
        xz.1 -1 327 379 603 0.0 0.0 0.0 1536.0 0.745
        yz.1 379 327 763 603 0.0 0.0 0.0 1536.0 0.745
        textures 763 166 863 603 0 0
```

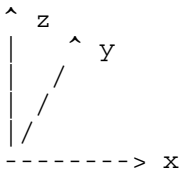
```
      object 763 52 863 166
   }

}

block 0 {
   ( 0.0 0.0 32.0 ) ( 0.0 500.0 32.0 ) ( 500.0 0.0 32.0 ) wood 0 0 0 1.00 1.00 : 0000 0.0 50.0 ndef ndef 0.0 0000
   ( 32.0 0.0 0.0 ) ( 32.0 0.0 500.0 ) ( 32.0 500.0 0.0 ) wood 0 0 0 1.00 1.00 : 0000 0.0 50.0 ndef ndef 0.0 0000
   ( 0.0 -32.0 0.0 ) ( 0.0 -32.0 500.0 ) ( 500.0 -32.0 0.0 ) wood 0 0 0 1.00 1.00 : 0000 0.0 50.0 ndef ndef 0.0 0000
   ( -32.0 0.0 0.0 ) ( -32.0 500.0 0.0 ) ( -32.0 0.0 500.0 ) wood 0 0 0 1.00 1.00 : 0000 0.0 50.0 ndef ndef 0.0 0000
   ( 0.0 32.0 0.0 ) ( 500.0 32.0 0.0 ) ( 0.0 32.0 500.0 ) wood 0 0 0 1.00 1.00 : 0000 0.0 50.0 ndef ndef 0.0 0000
   ( 0.0 0.0 -32.0 ) ( 500.0 0.0 -32.0 ) ( 0.0 500.0 -32.0 ) wood 0 0 0 1.00 1.00 : 0000 0.0 50.0 ndef ndef 0.0 0000
   {
      0000
      ndef
   }
}

level 0 {
   <block.$$w>
   <Palette.raw>
   30
   0.000
   60.000
   <test.wdl>
   100 100 100
   0 0 100
   100 0 0
   0 0 0
   19.60784340 19.60784340 19.60784340
   3.92156863 3.92156863 3.92156863
}

light 1 {
   49.84313583 49.84313583 49.84313583 300.0
}

object 0 {
   (0.0 0.0 0.0) (0.0 0.0 0.0) (1.0 1.0 1.0)
   object 1
   level 0
}

object 1 {
   (0.0 0.0 0.0) (0.0 0.0 0.0) (1.0 1.0 1.0)
   object 2, 3
}

object 2 {
   (-44.0 4.0 0.0) (0.0 0.0 0.0) (1.0 1.0 1.0)
   block 0
}

object 3 {
   (80.0 0.0 48.0) (1.570796 0.0 0.0) (1.0 1.0 1.0)
   light 1
}
```

All elements of the WMP file are surrounded by winged brackets { }. The WED element has no influence on the level, it just stores the palette of WED's texture and 3D display (which has only 256 colors) and the WED settings when the level was saved.

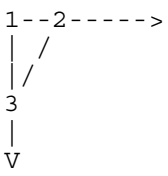
The important elements are the level properties, the blocks, lights, sounds, entities, and the objects that store the group information.

Blocks

A4 uses a right-handed XYZ coordinate system with the Z-axis standing upright. If you're not familiar with the reference of "right-handed" to a coordinate system, it basically provides a tactile and visual discription of the mechanics of the system. If you wrap your right hand around the z-axis, with your thumb facing the positive position, and clench your hand, your knuckles will face in the direction of the positive x-axis and your fingertips will face the postitive y-axis. Here's a picture of A4's coordinate system:



Blocks are the primary componens of a MAP file. Each block defines a solid or passable convex region. Blocks define this region as the intersection of four or more planes. Each plane is defined by three non-colinear points. These points must go in a clockwise orientation:



Each block statement looks like this:

```
block 0 {
   ( 0.0 0.0 32.0 ) ( 0.0 500.0 32.0 ) ( 500.0 0.0 32.0 ) wood 0 0 0 1.00 1.00 : 0000 0.0 50.0 ndef ndef 0.0 0000
   ( 32.0 0.0 0.0 ) ( 32.0 0.0 500.0 ) ( 32.0 500.0 0.0 ) wood 0 0 0 1.00 1.00 : 0000 0.0 50.0 ndef ndef 0.0 0000
   ( 0.0 -32.0 0.0 ) ( 0.0 -32.0 500.0 ) ( 500.0 -32.0 0.0 ) wood 0 0 0 1.00 1.00 : 0000 0.0 50.0 ndef ndef 0.0 0000
   ( -32.0 0.0 0.0 ) ( -32.0 500.0 0.0 ) ( -32.0 0.0 500.0 ) wood 0 0 0 1.00 1.00 : 0000 0.0 50.0 ndef ndef 0.0 0000
   ( 0.0 32.0 0.0 ) ( 500.0 32.0 0.0 ) ( 0.0 32.0 500.0 ) wood 0 0 0 1.00 1.00 : 0000 0.0 50.0 ndef ndef 0.0 0000
   ( 0.0 0.0 -32.0 ) ( 500.0 0.0 -32.0 ) ( 0.0 500.0 -32.0 ) wood 0 0 0 1.00 1.00 : 0000 0.0 50.0 ndef ndef 0.0 0000
   {
      0000
      ndef
   }
}
```

That's probably just a bit confusing when you first see it. It defines a cube that extends from (-32,-32,-32) to (32,32,32). Here's what a single line means:

```
 ( 0.0 0.0 32.0 ) ( 0.0 500.0 32.0 ) ( 500.0 0.0 32.0 )  wood      0     0      0    1.00    1.00 :
   1st point          2nd point          3rd point     texture  x_offs y_offs angle x_scale y_scale

  0000    0.0    50.0   ndef ndef 0.0 0000
 flags  ambient albedo  future expansion - not used yet
```

Here are more details about those fields:

1st point \ Those three points define a plane, so they must not be colinear.
2nd point > Each plane should only be defined once.
3rd point / Plane normal is oriented towards the cross product of (p1 - p2) and (p3 -p2)

Texture - name of the texture.
X_offs - texture x-offset in pixels
Y_offs - texture y-offset in pixels

Angle - texture rotation angle, in degree
X_scale - size of a pixel in x-direction (pixels per quant)
Y_scale - size of a pixel in y-direction (pixels per quant)

flags - render mode flags for the texture.
ambient - brightness value in percent
albedo - albedo/fog value in percent

Map Properties

The following structure contains the global level or map properties:

```
level 0 {
   <block.$$w>    - interim WAD
   <Palette.raw>  - palette
   30             - nexus
   0.000          - azimuth
   60.000         - elevation
   <test.wdl>     - script
   100 100 100    - fog 1
   0 0 100        - fog 2
   100 0 0        - fog 3
   0 0 0          - fog 4
   19.6 19.6 19.6 - sun
   3.9 3.9 3.9    - ambient
}
```

WAD – the name of the interim WAD file that contains all textures used in the level.
Palette – the name of the palette file. The .RAW format just consists of 768 bytes r,g,b values for the 256 colors.
Nexus – the nexus value, used for running and publishing the game.
Azimuth – the horizontal angle of the sun in degrees, counterclockwise beginning with 0 = East.
Elevation – the vertikal angle of the sun in degrees.
Script – the WDL script that runs the game
Fog 1..4 – the four fog colors in RGB percent
Sun – the strength and colour of the sun light in RGB percent
Ambient – the strength and colour of the ambient brightness in RGB percent

(to be continued)

Light and Sound Sources

Entities

Groups

The MDL model format

A wireframe mesh, made of triangles, gives the general shape of a model. 3D vertices define the position of triangles. For each triangle in the wireframe, there will be a corresponding triangle cut from the skin picture. Or, in other words, for each 3D vertex of a triangle that describes a XYZ position, there will be a corresponding 2D vertex positioned that describes a UV position on the skin picture.

It is not necessary that the triangle in 3D space and the triangle on the skin have the same shape (in fact, it is not possible for all triangles), but they should have shapes roughly similar, to limit distortion and aliasing. Several animation frames of a model are just several sets of 3D vertex positions. The 2D vertex positions always remain the same.

A MDL file contains:
- A list of skin textures in 8-bit palettized or 16-bit 5-6-5 format.
- A list of skin vertices, that are just the UV position of vertices on the skin texture.
- A list of triangles, which describe the general shape of the model.
- A list of animation frames. Each frame holds a list of 3D vertices.
- A list of bone vertices, which are used for creating the animation frames.

MDL file header

Once the file header is read, all the other model parts can be found just by calculating their position in the file. Here is the format of the .MDL file header:

```
typedef float vec3[3];

typedef struct {
  char version[4];   // "MDL3" or "MDL4"
  long final;        // not used yet
  vec3 scale;        // 3D position scale factors.
  vec3 offset;       // 3D position offset.
  float pad;         // not used yet.
  vec3 eye;          // not used yet.
  long numskins ;    // number of skin textures
  long skinwidth;    // width of skin texture; must be a multiple of 2
  long skinheight;   // height of skin texture
  long numverts;     // number of 3d wireframe vertices
  long numtris;      // number of triangles surfaces
  long numframes;    // number of frames
  long numskinverts; // number of 2D skin vertice
  long flags;        // 0 = normal, 1 = terrain model
  long numbones;     // number of bone vertices (MDL4 only, otherwise 0)
} mdl_header;
```

The size of this header is 0x54 bytes (84).

The "MDL3" format is used by the A4 engine, while the "MDL4" format is used by the A5 engine. After the file header follow the skins, the skin vertices, the triangles, the frames, and finally the bones (future expansion).

MDL skin format

The model skins are flat pictures that represent the texture that should be applied on the model. There can be more than one skin. You will find the first skin just after the model header, at offset `baseskin = 0x54`. There are `numskins` skins to read. Each of these model skins is either in 8-bit palettized (`type` == 0) or in 16-bit 5-6-5 format (`type` == 2). The structure is:

```
typedef byte unsigned char;
typedef struct {
  int skintype;    // 0 for 8 bit (bpp == 1), 2 for 16 bit (bpp == 2)
  byte skin[skinwidth*skinheight*bpp]; // the skin picture
} mdl_skin_t;
```

8 bit skins are a table of bytes, which represent an index in the level palette. If the model is rendered in overlay mode, index 0x00 indicates transparency. 16 bit skins are a table of shorts, which represent a true colour with the upper 5 bits for the red, the middle 6 bits for the green, and the lower 5 bits for the blue component. Green has one bit more because the human eye is more sensitive to green than to other colours. If the model is rendered in overlay mode, colour value 0x0000 indicates transparency.

The width of skins should be a multiple of 4, to ensure long word alignement. The skin pictures are usually made of as many pieces as there are independent parts in the model. For instance, for the a player, there may be two pieces that defines the body, and two others that define the gun.

MDL skin vertices

The list of skin vertices indicates only the position on texture picture, not the 3D position. That's because for a given vertex, the position on skin is constant, while the position in 3D space varies with the animation. The list of skin vertices is made of these structures:

```
typedef struct
{
  short u;  // position, horizontally in range 0..skinwidth-1
  short v;  // position, vertically in range 0..skinheight-1
} mdl_uvvert_t;

mdl_uvvert_t skinverts[numskinverts];
```

u and v are the pixel position on the skin picture. The skin vertices are stored in a list, that is stored at offset `basestverts = baseskin + skinsize`. `skinsize` is the sum of the size of all skin pictures. If they are all 8-bit skins, then `skinsize = (4 + skinwidth * skinheight) * numskins`. If they are 16-bit skins, then `skinsize = (4 + skinwidth * skinheight * 2) * numskins`.

MDL mesh triangles

The model wireframe mesh is made of a set of triangle facets, with vertices at the boundaries. Triangles should all be valid triangles, not degenerates (like points or lines). The triangle face must be pointing to the outside of the model. Only vertex indexes are stored in triangles. Here is the structure of triangles:

```
typedef struct {
   short index_xyz[3]; // Index of 3 3D vertices in range 0..numverts
   short index_uv[3];  // Index of 3 skin vertices in range 0..numskinverts
} mdl_triangle_t;

mdl_triangle_t triangles[numtris];
```

At offset `basetri = baseverts + numskinverts * sizeof(uvvert_t)` in the .MDL file you will find the triangle list.

MDL frames

A model contains a set of animation frames, which can be used in relation with the behavior of the modeled entity, so as to display it in various postures (walking, attacking, spreading its guts all over the place, etc). Basically the frame contains of vertex positions and normals. Because models can have ten thousands of vertices and hundreds of animation frames, vertex posistion are packed, and vertex normals are indicated by an index in a fixed table, to save disk and memory space.
Each frame vertex is defined by a 3D position and a normal for each of the 3D vertices in the model. In the MDL3 format, the vertices are always packed as bytes; in the MDL4 format that is used by the A5 engine they can also be packed as words (unsigned shorts). Therefore the MDL4 format allows more precise animation of huge models, and inbetweening with less distortion.

```
typedef struct {
  byte  rawposition[3];   // X,Y,Z coordinate, packed on 0..255
  byte  lightnormalindex; // index of the vertex normal
} mdl_trivertxb_t;

typedef struct {
  unsigned short rawposition[3];   // X,Y,Z coordinate, packed on 0..65536
  byte  lightnormalindex; // index of the vertex normal
  byte  boneindex;        // index of the bone this vertex belongs to
} mdl_trivertxs_t;
```

To get the real X coordinate from the packed coordinates, multiply the X coordinate by the X scaling factor, and add the X offset. Both the scaling factor and the offset for all vertices can be found in the `mdl_header` struct. The formula for calculating the real vertex positions is:

```
float position[i] = (scale[i] * rawposition[i] ) + offset[i];
```

The lightnormalindex field is an index to the actual vertex normal vector. This vector is the average of the normal vectors of all the faces that contain this vertex. The normal is necessary to calculate the Gouraud shading of the faces, but actually a crude estimation of the actual vertex normal is sufficient. That's why, to save space and to reduce the number of computations needed, it has been chosen to approximate each vertex normal.
The ordinary values of lightnormalindex are comprised between 0 and 161, and directly map into the index of one of the 162 precalculated normal vectors:

```
float lightnormals[162][3] = {
   {-0.525725, 0.000000, 0.850650}, {-0.442863, 0.238856, 0.864188}, {-0.295242, 0.000000, 0.955423},
   {-0.309017, 0.500000, 0.809017}, {-0.162460, 0.262866, 0.951056}, {0.000000, 0.000000, 1.000000},
   {0.000000, 0.850651, 0.525731}, {-0.147621, 0.716567, 0.681718}, {0.147621, 0.716567, 0.681718},
   {0.000000, 0.525731, 0.850651}, {0.309017, 0.500000, 0.809017}, {0.525731, 0.000000, 0.850651},
   {0.295242, 0.000000, 0.955423}, {0.442863, 0.238856, 0.864188}, {0.162460, 0.262866, 0.951056},
   {-0.681718, 0.147621, 0.716567}, {-0.809017, 0.309017, 0.500000}, {-0.587785, 0.425325, 0.688191},
   {-0.850651, 0.525731, 0.000000}, {-0.864188, 0.442863, 0.238856}, {-0.716567, 0.681718, 0.147621},
   {-0.688191, 0.587785, 0.425325}, {-0.500000, 0.809017, 0.309017}, {-0.238856, 0.864188, 0.442863},
   {-0.425325, 0.688191, 0.587785}, {-0.716567, 0.681718, -0.147621}, {-0.500000, 0.809017, -0.309017},
   {-0.525731, 0.850651, 0.000000}, {0.000000, 0.850651, -0.525731}, {-0.238856, 0.864188, -0.442863},
   {0.000000, 0.955423, -0.295242}, {-0.262866, 0.951056, -0.162460}, {0.000000, 1.000000, 0.000000},
   {0.000000, 0.955423, 0.295242}, {-0.262866, 0.951056, 0.162460}, {0.238856, 0.864188, 0.442863},
   {0.262866, 0.951056, 0.162460}, {0.500000, 0.809017, 0.309017}, {0.238856, 0.864188, -0.442863},
   {0.262866, 0.951056, -0.162460}, {0.500000, 0.809017, -0.309017}, {0.850651, 0.525731, 0.000000},
   {0.716567, 0.681718, 0.147621}, {0.716567, 0.681718, -0.147621}, {0.525731, 0.850651, 0.000000},
   {0.425325, 0.688191, 0.587785}, {0.864188, 0.442863, 0.238856}, {0.688191, 0.587785, 0.425325},
```

```
   {0.809017, 0.309017, 0.500000}, {0.681718, 0.147621, 0.716567}, {0.587785, 0.425325, 0.688191},
   {0.955423, 0.295242, 0.000000}, {1.000000, 0.000000, 0.000000}, {0.951056, 0.162460, 0.262866},
   {0.850651, -0.525731, 0.000000}, {0.955423, -0.295242, 0.000000}, {0.864188, -0.442863, 0.238856},
   {0.951056, -0.162460, 0.262866}, {0.809017, -0.309017, 0.500000}, {0.681718, -0.147621, 0.716567},
   {0.850651, 0.000000, 0.525731}, {0.864188, 0.442863, -0.238856}, {0.809017, 0.309017, -0.500000},
   {0.951056, 0.162460, -0.262866}, {0.525731, 0.000000, -0.850651}, {0.681718, 0.147621, -0.716567},
   {0.681718, -0.147621, -0.716567}, {0.850651, 0.000000, -0.525731}, {0.809017, -0.309017, -0.500000},
   {0.864188, -0.442863, -0.238856}, {0.951056, -0.162460, -0.262866}, {0.147621, 0.716567, -0.681718},
   {0.309017, 0.500000, -0.809017}, {0.425325, 0.688191, -0.587785}, {0.442863, 0.238856, -0.864188},
   {0.587785, 0.425325, -0.688191}, {0.688197, 0.587780, -0.425327}, {-0.147621, 0.716567, -0.681718},
   {-0.309017, 0.500000, -0.809017}, {0.000000, 0.525731, -0.850651}, {-0.525731, 0.000000, -0.850651},
   {-0.442863, 0.238856, -0.864188}, {-0.295242, 0.000000, -0.955423}, {-0.162460, 0.262866, -0.951056},
   {0.000000, 0.000000, -1.000000}, {0.295242, 0.000000, -0.955423}, {0.162460, 0.262866, -0.951056},
   {-0.442863,-0.238856, -0.864188}, {-0.309017,-0.500000, -0.809017}, {-0.162460, -0.262866, -0.951056},
   {0.000000, -0.850651, -0.525731}, {-0.147621, -0.716567, -0.681718}, {0.147621, -0.716567, -0.681718},
   {0.000000, -0.525731, -0.850651}, {0.309017, -0.500000, -0.809017}, {0.442863, -0.238856, -0.864188},
   {0.162460, -0.262866, -0.951056}, {0.238856, -0.864188, -0.442863}, {0.500000, -0.809017, -0.309017},
   {0.425325, -0.688191, -0.587785}, {0.716567, -0.681718, -0.147621}, {0.688191, -0.587785, -0.425325},
   {0.587785, -0.425325, -0.688191}, {0.000000, -0.955423, -0.295242}, {0.000000, -1.000000, 0.000000},
   {0.262866, -0.951056, -0.162460}, {0.000000, -0.850651, 0.525731}, {0.000000, -0.955423, 0.295242},
   {0.238856, -0.864188, 0.442863}, {0.262866, -0.951056, 0.162460}, {0.500000, -0.809017, 0.309017},
   {0.716567, -0.681718, 0.147621}, {0.525731, -0.850651, 0.000000}, {-0.238856, -0.864188, -0.442863},
   {-0.500000, -0.809017, -0.309017}, {-0.262866, -0.951056, -0.162460}, {-0.850651, -0.525731, 0.000000},
   {-0.716567, -0.681718, -0.147621}, {-0.716567, -0.681718, 0.147621}, {-0.525731, -0.850651, 0.000000},
   {-0.500000, -0.809017, 0.309017}, {-0.238856, -0.864188, 0.442863}, {-0.262866, -0.951056, 0.162460},
   {-0.864188, -0.442863, 0.238856}, {-0.809017, -0.309017, 0.500000}, {-0.688191, -0.587785, 0.425325},
   {-0.681718, -0.147621, 0.716567}, {-0.442863, -0.238856, 0.864188}, {-0.587785, -0.425325, 0.688191},
   {-0.309017, -0.500000, 0.809017}, {-0.147621, -0.716567, 0.681718}, {-0.425325, -0.688191, 0.587785},
   {-0.162460, -0.262866, 0.951056}, {0.442863, -0.238856, 0.864188}, {0.162460, -0.262866, 0.951056},
   {0.309017, -0.500000, 0.809017}, {0.147621, -0.716567, 0.681718}, {0.000000, -0.525731, 0.850651},
   {0.425325, -0.688191, 0.587785}, {0.587785, -0.425325, 0.688191}, {0.688191, -0.587785, 0.425325},
   {-0.955423, 0.295242, 0.000000}, {-0.951056, 0.162460, 0.262866}, {-1.000000, 0.000000, 0.000000},
   {-0.850651, 0.000000, 0.525731}, {-0.955423, -0.295242, 0.000000}, {-0.951056, -0.162460, 0.262866},
   {-0.864188, 0.442863, -0.238856}, {-0.951056, 0.162460, -0.262866}, {-0.809017, 0.309017, -0.500000},
   {-0.864188,-0.442863, -0.238856}, {-0.951056,-0.162460, -0.262866}, {-0.809017, -0.309017, -0.500000},
   {-0.681718, 0.147621, -0.716567}, {-0.681718, -0.147621, -0.716567}, {-0.850651, 0.000000, -0.525731},
   {-0.688191, 0.587785, -0.425325}, {-0.587785, 0.425325, -0.688191}, {-0.425325, 0.688191, -0.587785},
   {-0.425325,-0.688191, -0.587785}, {-0.587785,-0.425325, -0.688191}, {-0.688197,-0.587780, -0.425327}
};
```

A whole frame has the following structure:

```
typedef struct {
  long type;      // 0 for byte-packed positions, and 2 for word-packed positions
  mdl_trivertx_t bboxmin,bboxmax;   // bounding box of the frame
  char name[16];  // name of frame, used for animation
  mdl_trivertx_t vertex[numverts];  // array of vertices, either byte or short packed
} mdl_frame_t;
```

The size of each frame is `sizeframe = 20 + (numverts+2) * sizeof(mdl_trivertx_t)`, while `mdl_trivertx_t` is either `mdl_trivertxb_t` or `mdl_trivertxs_t`, depending on whether the type is 0 or 2. In the MDL3 format the type is always 0. The beginning of the frames can be found in the .MDL file at offset `baseframes = basetri + numtris * sizeof(mdl_triangle_t)`.

MDL bones

This is only for future expansion of the MDL format, and not implemented yet.

Bones are a linked list of 3D vertices that are used for animation in the MDL4 format. Each bone vertex can have a parent, and several childs. If a bone vertex is moved, the childs move with it. If on moving a bone vertex the connection line to his parent rotates, it's childs are rotated likewise about the parent position. If the distance of the bone vertex to its parent changes, the change is added onto the distance between childs and parent. So the movement of the childs is done in a spherical coordinate system, it is a combination of a rotation and a radius change.
Each bone vertex has an influence on one or more mesh vertices. The mesh vertices influenced by a bone vertex move the same way as it's childs. If a mesh vertex is influenced by several bone vertices, it is moved by the average of the bone's movement.