File format version 5

Format Specification

Contents

Chapter 1

Overview

This document describes version 5 of the OpenCTM file format.

1.1 File structure

The structure of an OpenCTM file is as follows:

[Header]
[Body data]

Each part of the file is described in the following chapters.

1.2 Data formats

All integer fields are stored in 32-bit little endian format (least significant byte first).

All floating point fields are stored in 32-bit binary IEEE 754 format (little endian).

All strings are stored as a 32-bit integer string length (number of bytes) followed by a UTF-8 format string (there is no zero termination and no BOM).

1.3 Packed data

Some portions of the file are be packed by the lossless LZMA entropy coder, and are encoded as follows:

Offset	Type	Description
0	Integer	Packed size (number of bytes, p).
4	-	LZMA specific props (five bytes, required by the LZMA decoder).
9	-	LZMA packed stream (p bytes long) that has been generated by the LzmaCompress() function of the LZMA API.

The length of the unpacked data is always known from the context (e.g. the triangle count uniquely defines the number of bytes required for the uncompressed triangle indices array).

1.3.1 Element interleaving

Some packed data arrays use element level interleaving, meaning that the data values are rearranged at the element level. For instance, in a data array with three elements per value (stride = 3), x, y and z, the elements are rearranged as follows:

$$x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_N, y_N, z_N \Rightarrow$$
$$x_1, x_2, \ldots, x_N, y_1, y_2, \ldots, y_N, z_1, z_2, \ldots, z_N$$

When decompressing an array that uses element interleaving, the process is reversed.

1.3.2 Byte interleaving

All packed data arrays use byte level interleaving, meaning that data values are rearranged at the byte level. For instance, in an integer array, where each element consists of four bytes: a, b, c and d, the bytes are rearranged as follows:

$$a_1, b_1, c_1, d_1, a_2, b_2, c_2, d_2, \ldots, a_N, b_N, c_N, d_N \Rightarrow$$
$$a_1, a_2, \ldots, a_N, b_1, b_2, \ldots, b_N, c_1, c_2, \ldots, c_N, d_1, d_2, \ldots, d_N$$

When decompressing an array that uses byte interleaving, the process is reversed.

1.3.3 Signed magnitude representation

Some packed integer arrays use signed magnitude representation.

A signed magnitude value, x', is converted to a two's complement value, x, with the following method:

$$x = \begin{cases} x' \; shr \; 1 & x'_0 = 0 \\ -((x'+1) \; shr \; 1) & x'_0 = 1 \end{cases}$$

...where x'_0 is the least significant bit of x'.

Chapter 2

Header

The file must start with a header, which looks as follows:

Offset	Type	Description
0	Integer	Magic identifier (0x4d54434f, or ”OCTM” when read as ASCII).
4	Integer	File format version (0x00000005 = version 5).
8	Integer	Compression method, which must be one of the following: 0x00574152 - Use the RAW compression method. 0x0031474d - Use the MG1 compression method. 0x0032474d - Use the MG2 compression method.
12	Integer	Vertex count.
16	Integer	Triangle count.
20	Integer	UV map count.
24	Integer	Attribute map count.
28	Integer	Boolean flags, or:ed together: 0x00000001 - The file contains per-vertex normals.
32	String	File comment (p bytes long string).

The length of the file header is $36 + p$ bytes, where p is the length of the comment string.

Chapter 3

Body data

The body data follows immediately after the file header. Its file offset is dictated by the length of the file header.

The format of the body data is specific for each compression method, which is defined by the ”Compression method” field in the header.

The body data contains the vertex, index, normal, UV map and attribute map data, usually in a compressed form.

3.1 RAW

The layout of the body data for the RAW compression method is:

[Indices]
[Vertices]
[Normals]
[UV map 0]
[UV map 1]
...
[UV map N]
[Attribute map 0]
[Attribute map 1]
...
[Attribute map M]

3.1.1 Indices

The indices are stored as an integer identifier, 0x58444e49 (”INDX”), followed by all the triangle indices. Each index is an unsigned integer value. There are three

indices per triangle, and the number of triangles is given by the ”Triangle count” field in the header:

Offset	Type	Description
0	Integer	Identifier (0x58444e49, or ”INDX” when read as ASCII).
4	Integer	Vertex index for the 1st corner of the 1st triangle.
8	Integer	Vertex index for the 2nd corner of the 1st triangle.
12	Integer	Vertex index for the 3rd corner of the 1st triangle.
16	Integer	Vertex index for the 1st corner of the 2nd triangle.
...		

The length of the indices section is $4(1 + 3N)$ bytes, where N is the triangle count.

3.1.2 Vertices

The vertices are stored as an integer identifier, 0x54524556 (”VERT”), followed by all the vertex coordinates. Each vertex coordinate is stored as three floating point values (x, y, z), and the number of vertices is given by the ”Vertex count” field in the header:

Offset	Type	Description
0	Integer	Identifier (0x54524556, or ”VERT” when read as ASCII).
4	Float	x coordinate of the 1st vertex.
8	Float	y coordinate of the 1st vertex.
12	Float	z coordinate of the 1st vertex.
16	Float	x coordinate of the 2nd vertex.
...		

The length of the vertices section is $4(1 + 3N)$ bytes, where N is the vertex count.

3.1.3 Normals

The normals section is optional, and only present if the per-vertex normals flag is set in the header.

The normals are stored as an integer identifier, 0x4d524f4e (”NORM”), followed by all the normal coordinates. Each normal is stored as three floating point values (x, y, z), and the number of normals is given by the ”Vertex count” field in the header:

Offset	Type	Description
0	Integer	Identifier (0x4d524f4e, or ”NORM” when read as ASCII).
4	Float	x coordinate of the 1st normal.
8	Float	y coordinate of the 1st normal.
12	Float	z coordinate of the 1st normal.
16	Float	x coordinate of the 2nd normal.
...		

The length of the normals section is $4(1 + 3N)$ bytes, where N is the vertex count.

3.1.4 UV maps

There can be zero or more UV maps. The number of UV maps is given by the UV map count in the header.

Each UV map starts with an integer identifier, 0x43584554 (”TEXC”), followed by two strings (the UV map name and the UV map file name reference), and finally all the UV coordinates. Each UV coordinate is stored as two floating point values (u, v), and the number of UV coordinates is given by the ”Vertex count” field in the header:

Offset	Type	Description
0	Integer	Identifier (0x43584554, or ”TEXC” when read as ASCII).
4	String	Unique UV map name (p bytes long string).
$8 + p$	String	UV map file name reference (q bytes long string).
$12 + p + q$	Float	u coordinate of the 1st UV coordinate.
$16 + p + q$	Float	v coordinate of the 1st UV coordinate.
$20 + p + q$	Float	u coordinate of the 2nd UV coordinate.
...		

The length of a UV map section is $4(3 + 2N) + p + q$ bytes, where N is the vertex count, p is the name string length, and q is the file name reference string length.

3.1.5 Attribute maps

There can be zero or more attribute maps. The number of attribute maps is given by the attribute map count in the header.

Each attribute map starts with an integer identifier, 0x52545441 (”ATTR”), followed by the attribute map name string, and finally all the attribute values. Each attribute value is stored as four floating point values (a, b, c, d), and the number of attribute values is given by the ”Vertex count” field in the header:

Offset	Type	Description
0	Integer	Identifier (0x52545441, or "ATTR" when read as ASCII).
4	String	Unique attribute map name (p bytes long string).
$8+p$	Float	a component of the 1st attribute value.
$12+p$	Float	b component of the 1st attribute value.
$16+p$	Float	c component of the 1st attribute value.
$20+p$	Float	d component of the 1st attribute value.
$24+p$	Float	a component of the 2nd attribute value.
...		

The length of an attribute map section is $4(2+4N)+p$ bytes, where N is the vertex count, and p is the name string length.

3.2 MG1

The layout of the body data for the MG1 compression method is:

[Indices]
[Vertices]
[Normals]
[UV map 0]
[UV map 1]
...
[UV map N]
[Attribute map 0]
[Attribute map 1]
...
[Attribute map M]

3.2.1 Indices

The triangle indices are stored as an integer identifier, 0x58444e49 ("INDX"), followed by a packed integer array with element interleaving (see 1.3).

Offset	Type	Description
0	Integer	Identifier (0x58444e49, or "INDX" when read as ASCII).
4	-	Packed indices data.

The unpacked indices array contains delta-encoded indices:

$i'_{1,1}, i'_{1,2}, i'_{1,3}, i'_{2,1}, i'_{2,2}, i'_{2,3} \ldots, i'_{M,1}, i'_{M,2}, i'_{M,3}$

...that translate into the original triangle indices with the following method:

$$i_{k,1} = \begin{cases} i'_{k,1} + i_{k-1,1} & (k \geq 2) \\ i'_{k,1} & (k = 1) \end{cases}$$

$$i_{k,2} = \begin{cases} i'_{k,2} + i_{k-1,2} & (k \geq 2, i_{k,1} = i_{k-1,1}) \\ i'_{k,2} + i_{k,1} & (\text{otherwise}) \end{cases}$$

$$i_{k,3} = i'_{k,3} + i_{k,1}$$

...where $i_{k,1}, i_{k,2}$ and $i_{k,3}$ are the 1:st, 2:nd and 3:rd vertex index of the k:th triangle, respectively.

Please note that the indices should be sorted in such a manner that $i'_{k,1} \geq 0, i'_{k,2} \geq 0$ and $i'_{k,3} \geq 0 \; \forall \; k$.

3.2.2 Vertices

The vertices are stored as an integer identifier, 0x54524556 (”VERT”), followed by a packed float array without element interleaving (see 1.3).

Offset	Type	Description
0	Integer	Identifier (0x54524556, or ”VERT” when read as ASCII).
4	-	Packed vertices data.

The unpacked vertex array is stored as in the RAW format (x, y, z).

3.2.3 Normals

The normals section is optional, and only present if the per-vertex normals flag is set in the header.

The normals are stored as an integer identifier, 0x4d524f4e (”NORM”), followed by a packed float array with element interleaving (see 1.3).

Offset	Type	Description
0	Integer	Identifier (0x4d524f4e, or ”NORM” when read as ASCII).
4	-	Packed normals data.

The unpacked normal array is stored as in the RAW format (x, y, z).

3.2.4 UV maps

There can be zero or more UV maps. The number of UV maps is given by the UV map count in the header.

Each UV map starts with an integer identifier, 0x43584554 (”TEXC”), followed by two strings (the UV map name and the UV map file name reference), and finally the packed UV coordinate data.

The UV coordinate data is a packed float array with element interleaving (see 1.3).

Offset	Type	Description
0	Integer	Identifier (0x43584554, or ”TEXC” when read as ASCII).
4	String	Unique UV map name (p bytes long string).
$8 + p$	String	UV map file name reference (q bytes long string).
$12 + p + q$	-	Packed UV coordinate data.

...where p is the name string length, and q is the file name reference string length.

The unpacked UV coordinate array is stored as in the RAW format (u, v).

3.2.5 Attribute maps

There can be zero or more attribute maps. The number of attribute maps is given by the attribute map count in the header.

Each attribute map starts with an integer identifier, 0x52545441 (”ATTR”), followed by the attribute map name string, and finally the packed attribute values.

The attribute value data is a packed float array with element interleaving (see 1.3).

Offset	Type	Description
0	Integer	Identifier (0x52545441, or ”ATTR” when read as ASCII).
4	String	Unique attribute map name (p bytes long string).
$8 + p$	-	Packed attribute value data.

...where p is the name string length.

The unpacked attribute value array is stored as in the RAW format (a, b, c, d).

3.3 MG2

The layout of the body data for the MG2 compression method is:

[MG2 header]
[Vertices]
[Grid indices]
[Indices]
[Normals]
[UV map 0]
[UV map 1]
...
[UV map N]
[Attribute map 0]
[Attribute map 1]
...
[Attribute map M]

3.3.1 MG2 vertex coordinate coding

In the MG2 compression method, all the vertices are divided into a 3D grid, which can be described by an axis aligned bounding box (minimum fit to the vertices), and the division factors along the x, y and z axes, as shown in figure 3.1.

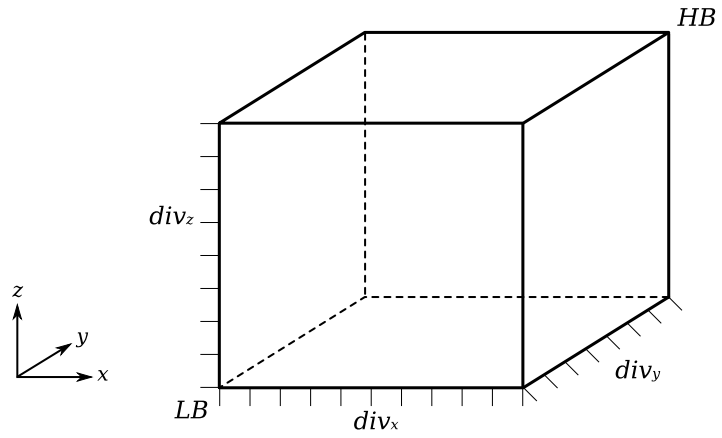

Figure 3.1: 3D space subdivision grid. LB and HB are the lower and higher bounds of the axis aligned bounding box. div_x, div_y and div_z are the number of divisions along each of the axes.

The vertices are all coded relative to the grid origin of the grid box to which they belong, and all vertices are associated with a grid box with a unique grid index.

The grid index, gi, is encoded as:

$gi = g_x + div_x(g_y + div_y \times g_z)$

...where g_x, g_y and g_z are the integer x, y and z positions of the grid box, within the grid, and:

$g_x \in [0, div_x), g_y \in [0, div_y), g_z \in [0, div_z)$

The grid box origin (lower bound) of each grid box is defined by:

$gridorigin_x(g_x) = LB_x + \frac{HB_x - LB_x}{div_x} g_x$

$gridorigin_y(g_y) = LB_y + \frac{HB_y - LB_y}{div_y} g_y$

$gridorigin_z(g_z) = LB_z + \frac{HB_z - LB_z}{div_z} g_z$

3.3.2 MG2 header

The MG2 header contains information about how to interpret the mesh data. The header looks as follows:

Offset	Type	Description
0	Integer	Identifier (0x4832474d, or ”MG2H” when read as ASCII).
4	Float	Vertex precision.
8	Float	Normal precision.
12	Float	LB_x (z coordinate of the lower bound of the bounding box).
16	Float	LB_y (y coordinate of the lower bound of the bounding box).
20	Float	LB_z (z coordinate of the lower bound of the bounding box).
24	Float	HB_x (x coordinate of the higher bound of the bounding box).
28	Float	HB_y (y coordinate of the higher bound of the bounding box).
32	Float	HB_z (z coordinate of the higher bound of the bounding box).
36	Integer	div_x (number of grid divisions along the x axis, ≥ 1).
40	Integer	div_y (number of grid divisions along the y axis, ≥ 1).
44	Integer	div_z (number of grid divisions along the z axis, ≥ 1).

3.3.3 Vertices

The vertices are stored as an integer identifier, 0x54524556 (”VERT”), followed by the packed vertex coordinate data.

The vertex coordinate data is a packed integer array with element interleaving (see 1.3).

Offset	Type	Description
0	Integer	Identifier (0x54524556, or ”VERT” when read as ASCII).
4	-	Packed vertex coordinate data.

The unpacked vertex array has three elements per vertex:

$x'_1, y'_1, z'_1, x'_2, y'_2, z'_2, ..., x'_N, y'_N, z'_N$

The original vertex coordinate, (x_k, y_k, z_k), for vertex number k is defined as:

$$dx_k = \begin{cases} x'_k + dx_{k-1} & (k \geq 2, gi_k = gi_{k-1}) \\ x'_k & (otherwise) \end{cases}$$

$$x_k = s \times dx_k + gridorigin_x(gi_k)$$

$$y_k = s \times y'_k + gridorigin_y(gi_k)$$

$$z_k = s \times z'_k + gridorigin_z(gi_k)$$

...where s is the vertex precision, gi_k is the k:th grid index (see 3.3.4), and $gridorigin(gi_k)$ is the origin (lower bound) of the grid box that is indicated by grid index gi_k, according to 3.3.1.

Please note that the vertices should be sorted in such a manner that $x'_k \geq 0, y'_k \geq 0$ and $z'_k \geq 0 \; \forall \; k$.

3.3.4 Grid indices

The grid indices are stored as an integer identifier, 0x58444947 (”GIDX”), followed by a packed integer array (see 1.3).

Offset	Type	Description
0	Integer	Identifier (0x58444947, or ”GIDX” when read as ASCII).
4	-	Packed grid indices data.

The unpacked grid indices array has one element per vertex:

$gi'_1, gi'_2, ..., gi'_N$

The grid index for vertex number k is defined as:

$$gi_k = \begin{cases} gi'_k + gi_{k-1} & (k \geq 2) \\ gi'_k & (k = 1) \end{cases}$$

Please note that the vertices should be sorted in such a manner that $gi'_k \geq 0 \, \forall \, k$.

3.3.5 Indices

The triangle indices are stored exactly as in the MG1 method (see 3.2.1).

3.3.6 Normals

The normals section is optional, and only present if the per-vertex normals flag is set in the header.

The normals are stored as an integer identifier, 0x4d524f4e (”NORM”), followed by a packed integer array with element interleaving (see 1.3).

Offset	Type	Description
0	Integer	Identifier (0x4d524f4e, or ”NORM” when read as ASCII).
4	-	Packed normals data.

Note: This section of the document is not yet complete... Please see the source code file compressMG2.c for more information about how to interpret the normal data array.

3.3.7 UV maps

There can be zero or more UV maps. The number of UV maps is given by the UV map count in the header.

Each UV map starts with an integer identifier, 0x43584554 (”TEXC”), followed by two strings (the UV map name and the UV map file name reference), the UV coordinate precision (a float value), and finally the packed UV coordinate data.

The UV coordinate data is a packed integer array with element interleaving and signed magnitude format (see 1.3).

Offset	Type	Description
0	Integer	Identifier (0x43584554, or "TEXC" when read as ASCII).
4	String	Unique UV map name (p bytes long string).
$8 + p$	String	UV map file name reference (q bytes long string).
$12 + p + q$	Float	UV coordinate precision, s.
$16 + p + q$	-	Packed UV coordinate data.

...where p is the name string length, and q is the file name reference string length.

The unpacked UV coordinate array contains delta-encoded coordinates:

$u'_1, v'_1, u'_2, v'_2, ..., u'_N, v'_N$

The original UV coordinates are restored with the following method:

$$u_k = \begin{cases} s \times (u'_k + u_{k-1}) & (k \geq 2) \\ s \times u'_k & (k = 1) \end{cases}$$

$$v_k = \begin{cases} s \times (v'_k + v_{k-1}) & (k \geq 2) \\ s \times v'_k & (k = 1) \end{cases}$$

...where s is the UV coordinate precision.

3.3.8 Attribute maps

There can be zero or more attribute maps. The number of attribute maps is given by the attribute map count in the header.

Each attribute map starts with an integer identifier, 0x52545441 ("ATTR"), followed by the attribute map name string, the attribute value precision (a float value), and finally the packed attribute values.

The attribute value data is a packed integer array with element interleaving and signed magnitude format (see 1.3).

Offset	Type	Description
0	Integer	Identifier (0x52545441, or "ATTR" when read as ASCII).
4	String	Unique attribute map name (p bytes long string).
$8 + p$	Float	Attribute value precision, s.
$12 + p$	-	Packed attribute value data.

...where p is the name string length.

The unpacked attribute value array contains delta-encoded attribute values:

$a'_1, b'_1, c'_1, d'_1, a'_2, b'_2, c'_2, d'_2, ..., a'_N, b'_N, c'_N, d'_N$

The original attributes are restored with the following method:

$$a_k = \begin{cases} s \times (a'_k + a_{k-1}) & (k \geq 2) \\ s \times a'_k & (k = 1) \end{cases}$$

$$b_k = \begin{cases} s \times (b'_k + b_{k-1}) & (k \geq 2) \\ s \times b'_k & (k = 1) \end{cases}$$

$$c_k = \begin{cases} s \times (c'_k + c_{k-1}) & (k \geq 2) \\ s \times c'_k & (k = 1) \end{cases}$$

$$d_k = \begin{cases} s \times (d'_k + d_{k-1}) & (k \geq 2) \\ s \times d'_k & (k = 1) \end{cases}$$

...where s is the attribute value precision.