

OpenEXR File Layout

Last Update: 04/24/07

This document gives an overview of the layout of OpenEXR image files as byte sequences. The text assumes that the reader is familiar with OpenEXR terms such as "channel", "attribute" or "data window". For an explanation of those terms see the Technical Introduction to OpenEXR.

This document does not define the OpenEXR file format. OpenEXR is defined as the file format that is read and written by the IlmImf open-source C++ library. If this document and the IlmImf library disagree, then the library takes precedence.

Table of Contents

Basic Data Types.....	2
Integers.....	2
Floating-point numbers	2
Text.....	2
Packing.....	3
File Layout.....	3
High-Level Layout.....	3
Magic Number.....	3
Version Field.....	3
Header.....	4
Scan Line Blocks.....	4
Line Offset Table.....	5
Tiles.....	6
Tile Offset Table.....	6
Predefined Attribute Types.....	6
Sample File.....	8

Basic Data Types

An OpenEXR file is a sequence of 8-bit bytes. Groups of bytes represent basic objects such as integral numbers, floating-point numbers and text. Those objects are grouped together to form compound objects such as attributes or scan lines.

Integers

Binary integral numbers with 8, 16, 32 or 64 bits are stored as 1, 2, 4 or 8 bytes. Integral numbers can be signed or unsigned. Signed numbers are represented using two's complement. Integral numbers are little-endian, that is, the least significant byte is closest to the start of the file.

OpenEXR uses the following six integer data types:

name	signed	size bytes	in
unsigned char	no		1
short	yes		2
unsigned short	no		2
int	yes		4
unsigned int	no		4
unsigned long	no		8

Floating-point numbers

Binary floating-point numbers with 16, 32 or 64 bits are stored as 2, 4 or 8 bytes. The representation of 32-bit and 64-bit floating-point numbers conforms to the IEEE 754 standard. The representation of 16-bit floating-point numbers is analogous to IEEE 754, but with 5 exponent bits and 10 bits for the fraction. The exponent bias is 15. Floating-point numbers are little-endian: the least significant bits of the fraction are in the byte closest to the beginning of the file, while the sign bit and the most significant bits of the exponent are in the byte closest to the end of the file.

The following table lists the names and sizes of OpenEXR's floating-point data types:

name	Size bytes	in
half		2
float		4
double		8

Text

Text strings are represented as sequences of 1-byte characters of type `char`. Depending on the context, either the end of a string is indicated by a null character (0x00), or the length of the string is indicated by an `int` that precedes the string.

Packing

Data in an OpenEXR file are densely packed; the file contains no "padding". For example, consider the following C struct:

```
struct SI
{
    short s;
    int i;
};
```

on most computers, the in-memory representation an `SI` object occupies eight bytes: 2 bytes for `s`, 2 padding bytes to ensure four-byte alignment of `i`, and 4 bytes for `i`. In an OpenEXR file the same object would consume only six bytes: 2 bytes for `s` and 4 bytes for `i`. The two padding bytes are not stored in the file.

File Layout

High-Level Layout

Depending on whether the pixels in an OpenEXR file are stored as scan lines or as tiles, the file consists of the following components:

file with scan lines:

magic number

version field

header

line offset table

scan line blocks

file with tiles:

magic number

version field

header

tile offset table

tiles

Magic Number

The magic number, of type `int`, is always 20000630 (decimal). It allows file readers to distinguish OpenEXR files from other files, since the first four bytes of an OpenEXR file are always 0x76, 0x2f, 0x31 and 0x01.

Version Field

The version field, of type `int`, is treated as two separate bit fields. The 8 least significant bits (bits 0 through 7) contain the file format version number. The 24 most significant bits (8 through 31) are treated as a set of boolean flags.

The current OpenEXR version number is 2. (Version 1 was used internally by ILM before OpenEXR was released as open source. The `IlmImf` library can no longer read or write version 1 files.)

Bit number 9 of the version field (bit mask 0x200) indicates how the pixels in the file are stored. If the bit is zero, the pixels are stored as scan lines; if the bit is one, the pixels are stored as tiles. The remaining 23 flags in the version field are currently unused and should be set to zero.

This means that currently there are only two valid settings for the version number field: 0x2 for scan-line based images, and 0x202 for tiled images.

Header

The header is a sequence of attributes, followed by a single null byte (0x00). The layout of an attribute is as follows:

attribute name
attribute type
attribute size
attribute value

The attribute name and the attribute type are null-terminated text strings. Excluding the null byte, the name and type must each be at least 1 byte and at most 31 bytes long.

The attribute size, of type `int`, indicates the size, in bytes, of the attribute value.

The layout of the attribute value depends on the attribute type. The `IlmImf` library predefines several different attribute types (see page 6). Application programs can define and store additional attribute types.

The header of every OpenEXR file must contain at least the following attributes:

attribute name	attribute type
<code>channels</code>	<code>chlist</code>
<code>compression</code>	<code>compression</code>
<code>dataWindow</code>	<code>box2i</code>
<code>displayWindow</code>	<code>box2i</code>
<code>lineOrder</code>	<code>lineOrder</code>
<code>pixelAspectRatio</code>	<code>float</code>
<code>screenWindowCenter</code>	<code>v2f</code>
<code>screenWindowWidth</code>	<code>float</code>

In addition, every tiled file must contain a tile description attribute, with name `"tiles"` and type `"tiledesc"`. The tile description attribute determines the size of the tiles and the number of resolution levels in the file.

The `IlmImf` library ignores tile description attributes in scan-line based files. The decision whether the file contains scan lines or tiles is based on the file's version field, not on the presence of a tile description attribute.

Scan Line Blocks

One or more scan lines are stored together as a scan-line block. The number of scan lines per block depends on how the pixel data are compressed:

compression method	number of scan lines per block
NO_COMPRESSION	1
RLE_COMPRESSION	1
ZIPS_COMPRESSION	1
ZIP_COMPRESSION	16
PIZ_COMPRESSION	32
PXR24_COMPRESSION	16
B44_COMPRESSION	32
B44A_COMPRESSION	32

Each scan line block has a y coordinate of type `int`. The block's y coordinate is equal to the pixel space y coordinate of the top scan line in the block. The top scan line block in the image is aligned with the top edge of the data window, that is, the y coordinate of the top scan line block is equal to the data window's minimum y.

If the height of the image's data window is not a multiple of the number of scan lines per block, then the block that contains the bottom scan line contains fewer scan lines than the other blocks.

The layout of a scan line block is as follows:

```

y coordinate
pixel data size
pixel data

```

The pixel data size, of type `int`, indicates the number of bytes occupied by the actual pixel data.

Within the pixel data, scan lines are stored top to bottom. Each scan line is contiguous, and within a scan line the data for each channel are contiguous. Channels are stored in alphabetical order, according to channel names. Within a channel, pixels are stored left to right.

If the file's compression method is `NO_COMPRESSION`, then the original, uncompressed pixel data are stored directly in the file. Otherwise, the uncompressed pixels are fed to the appropriate compressor, and either the compressed or the uncompressed data are stored in the file, whichever is smaller.

The layout of the compressed data depends on which compression method was applied. The compressed formats are not described here. For information on the compressed data formats, see the source code for the `IlmImf` library.

Line Offset Table

The line offset table allows random access to scan line blocks. The table is a sequence of scan line offsets, with one offset per scan line block. A scan line offset, of type `unsigned long`, indicates the distance, in bytes, between the start of the file and the start of the scan line block. In the table, scan line offsets are ordered according to increasing scan line y coordinates.

Tiles

The layout of a tile is as follows:

tile coordinates
pixel data size
pixel data

The tile coordinates, a sequence of four `ints`, `tileX`, `tileY`, `levelX`, `levelY` indicate the tile's position and resolution level. The pixel data size, of type `int`, indicates the number of bytes occupied by the pixel data.

The pixel in a tile data are laid out in the same way as in a scan line block, but the length of the scan lines is equal to the width of the tile, and the number of scan lines is equal to the height of the tile.

If the width of a resolution level is not a multiple of the file's tile width, then the tiles at the right edge of that resolution level have shorter scan lines. Similarly, if the height of a resolution level is not a multiple of the file's tile height, then tiles at the bottom edge of the resolution level have fewer scan lines.

Tile Offset Table

The tile offset table allows random access to tiles. The table is a sequence of tile offsets, one offset per tile. A tile offset, of type `unsigned long`, indicates the distance, in bytes, between the start of the file and the start of the tile. In the table scan line offsets are sorted the same way as tiles in `INCREASING_Y` order.

Predefined Attribute Types

The `IlmImf` library predefines the following attribute types:

type name	data
box2i	Four <code>ints</code> : <code>xMin</code> , <code>yMin</code> , <code>xMax</code> , <code>yMax</code>
box2f	Four <code>floats</code> : <code>xMin</code> , <code>yMin</code> , <code>xMax</code> , <code>yMax</code>
chlist	A sequence of channels followed by a null byte (0x00).
Channel layout:	
name	zero-terminated string, from 1 to 31 bytes long
pixel type	<code>int</code> , possible values are <code>UINT = 0</code> <code>HALF = 1</code> <code>FLOAT = 2</code>
pLinear	<code>unsigned char</code> , possible values are 0 and 1
reserved	three <code>chars</code> , should be zero
xSampling	<code>int</code>
ySampling	<code>int</code>

chromaticities	Eight floats: redX, redY, greenX, greenY, blueX, blueY, whiteX, whiteY
compression	unsigned char, possible values are NO_COMPRESSION = 0 RLE_COMPRESSION = 1 ZIPS_COMPRESSION = 2 ZIP_COMPRESSION = 3 PIZ_COMPRESSION = 4 PXR24_COMPRESSION = 5 B44_COMPRESSION = 6 B44A_COMPRESSION = 7
double	double
envmap	unsigned char, possible values are ENVMAP_LATLONG = 0 ENVMAP_CUBE = 1
float	float
int	int
keycode	Seven ints: filmMfcCode, filmType, prefix, count, perfOffset, perfsPerFrame, perfsPerCount
lineOrder	unsigned char, possible values are INCREASING_Y = 0 DECREASING_Y = 1 RANDOM_Y = 2
m33f	9 floats
m44f	16 floats
preview	Two unsigned ints: width, height, followed by 4×width×height unsigned chars of pixel data Scan lines are stored top to bottom, within a scan line pixels are stored from left to right. A pixel consists of four unsigned chars, R, G, B, A.
rational	An int, followed by an unsigned int.
string	String length, of type int, followed by a sequence of chars.
tilDESC	Two unsigned ints: xSize, ySize, followed by mode, of type unsigned char, where mode = levelMode + roundingMode×16 Possible values for levelMode: ONE_LEVEL = 0 MIPMAP_LEVELS = 1 RIPMAP_LEVELS = 2 Possible values for roundingMode: ROUND_DOWN = 0 ROUND_UP = 1
timecode	Two unsigned ints: timeAndFlags, userData

v2i	Two ints
v2f	Two floats
v3i	Three ints.
v3f	Three floats.

Sample File

The following is an annotated byte-by-byte listing of a complete OpenEXR file. The file contains a scan-line based image with four by three pixels. The image has two channels: G, of type HALF, and Z, of type FLOAT. The pixel data are not compressed. The entire file is 415 bytes long.

The first line of text in each of the gray boxes below lists up to 16 bytes of the file in hexadecimal notation. The second line in each box shows how the bytes are grouped into integers, floating-point numbers and text strings. The third and fourth lines indicate how those basic objects form compound objects such as attributes or the line offset table.

76	2f	31	01	02	00	00	00	63	68	61	6e	6e	65	6c	73
20000630				2				c h a n n e l s							
magic number				version, flags				attribute name							
								start of header							

00	63	68	6c	69	73	74	00	25	00	00	00	47	00	01	00
\0	c	h	l	i	s	t	\0	37				G	\0	HALF	
attribute type							attribute size				attribute value				

00	00	00	00	00	00	01	00	00	00	01	00	00	00	5a	00
		0	0				1				1				z \0

02	00	00	00	00	00	00	00	01	00	00	00	01	00	00	00
FLOAT					0		0		1				1		

00	63	6f	6d	70	72	65	73	73	69	6f	6e	00	63	6f	6d
\0	c	o	m	p	r	e	s	s	i	o	n	\0	c	o	m
attribute name												attribute type			


```

70 72 65 73 73 69 6f 6e 00 01 00 00 00 00 64 61
p r e s s i o n \0 | 1 | NONE | d a
| attribute size |value|

```

```

74 61 57 69 6e 64 6f 77 00 62 6f 78 32 69 00 10
t a W i n d o w \0 | b o x 2 i \0 |
attribute name | attribute type |

```

```

00 00 00 00 00 00 00 00 00 00 00 03 00 00 00 02
16 | 0 | 0 | 3 |
attribute size | attribute value

```

```

00 00 00 64 69 73 70 6c 61 79 57 69 6e 64 6f 77
2 | d i s p l a y W i n d o w
| attribute name

```

```

00 62 6f 78 32 69 00 10 00 00 00 00 00 00 00 00
\0 | b o x 2 i \0 | 16 | 0 |
| attribute type | attribute size | attribute value

```

```

00 00 00 03 00 00 00 02 00 00 00 6c 69 6e 65 4f
0 | 3 | 2 | l i n e O
| attribute name

```

```

72 64 65 72 00 6c 69 6e 65 4f 72 64 65 72 00 01
r d e r \0 | l i n e O r d e r \0 |
| attribute type |

```

```

00 00 00 00 70 69 78 65 6c 41 73 70 65 63 74 52
1 | INCY | p i x e l A s p e c t R
attribute size |value| attribute name

```

61	74	69	6f	00	66	6c	6f	61	74	00	04	00	00	00	00
a	t	i	o	\0		f	l	o	a	t	\0		4		
										attribute type		attribute size			

00	80	3f	73	63	72	65	65	6e	57	69	6e	64	6f	77	43
1.0				s c r e e n W i n d o w C											
attribute value				attribute name											

65	6e	74	65	72	00	76	32	66	00	08	00	00	00	00	00
e	n	t	e	r	\0		v	2	f	\0		8			
										attribute type		attribute size			

00	00	00	00	00	00	73	63	72	65	65	6e	57	69	6e	64
0.0		0.0				s c r e e n W i n d									
attribute value						attribute name									

6f	77	57	69	64	74	68	00	66	6c	6f	61	74	00	04	00
o	w	W	i	d	t	h	\0		f	l	o	a	t	\0	
										attribute type					

00	00	00	00	80	3f	00	3f	01	00	00	00	00	00	00	5f
4		1.0				\0		319							
size		attribute value						offset of scan line 0							
end of header										start of scan line offset table					

01	00	00	00	00	00	00	7f	01	00	00	00	00	00	00	00
351							383								
offset of scan line 1							offset of scan line 2								
end of scan line offset table															

00	00	00	18	00	00	00	00	00	54	29	d5	35	e8	2d	5c
0		24				0.000		0.042		0.365		0.092			
y		pixel data size				pixel data for G channel									
scan line 0															

28	81	3a	cf	e1	34	3e	8b	0b	bb	3d	89	74	f9	3e	01								
0.000985395						0.176643						0.0913306						0.487217					
pixel data for Z channel																							

00	00	00	18	00	00	00	37	38	76	33	74	3b	73	38	7f																				
1						24						0.527						0.233						0.932						0.556					
y						pixel data size						pixel data for G channel																							
scan line 1																																			

ab	e8	3e	8a	cf	54	3f	5b	6c	11	3f	20	35	50	3d	02								
0.454433						0.831292						0.56806						0.0508319					
pixel data for Z channel																							

00	00	00	18	00	00	00	23	3a	0a	34	02	3b	5d	3b	38																				
2						24						0.767						0.252						0.876						0.920					
y						pixel data size						pixel data for G channel																							
scan line 2																																			

f3	9a	3c	4d	ad	98	3e	1c	14	08	3f	4c	f3	03	3f						
0.0189148						0.298197						0.531557						0.515431		
pixel data for Z channel																				
end of file																				