WIKIPEDIA

ILBM

Interleaved Bitmap (**ILBM**) is an image file format conforming to the Interchange File Format (IFF) standard. The format originated on the Amiga platform, and on IBM-compatible systems, files in this format or the related PBM (Planar Bitmap) format are typically encountered in games from late 1980s and early 1990s that were either Amiga ports or had their graphical assets designed on Amiga machines.

A characteristic feature of the format is that it stores bitmaps in the form of interleaved bit planes, which gives the format its name; this reflects the way the Amiga graphics hardware natively reads graphics data from memory. A simple form of compression is supported to make ILBM files more compact.[4]

On the Amiga, these files are not associated with a particular file extension, though as they started being used on PC systems where extensions are systematically used, they employed a *.lbm* or occasionally a *.bbm* extension.

"ILBM" IFF Interleaved Bitmap

Filename extension	`.iff, .lbm`
Developed by	Electronic Arts
Initial release	14 January 1985
Type of format	Image file formats
Contained by	Interchange File Format
Standard	EA IFF 85: Standard for Interchange Format[1][2][3]
Open format?	Public domain source code

Contents

File format

ILBM is an implementation of the IFF file format consisting of a number of consecutive chunks, whose order can, to some extent, be varied. Each chunk has a different function and has the same basic format. This means that a program does not have to read or decode every chunk in a file, only the ones it wants to deal with or the ones it can understand.[4]

ILBM files usually contain enough information to allow them to be displayed by an image editing program, including image dimensions, palette and pixel data. Some files were designed to act as palettes for paint programs (pixel data left blank) or to be merged into another image. This makes them much more flexible, but also much more complex than other formats such as BMP.

For ILBMs the **BMHD** chunk and any other 'vital' chunks must appear before the **BODY** chunk. Any chunks appearing after **BODY** are considered 'extra' and many programs will leave them unread and unchanged.[4]

Type	Name	Description
FOURCC	chunkID	"FORM"
UINT32BE	lenChunk	Length of chunk data, in bytes. Does not include the pad byte. Will be the same as the file size minus eight bytes (this field and `chunkID` are not included in the count)
FOURCC	formatID	"ILBM" or "PBM "
BYTE[lenChunk - 12]	content	Actual data of the chunk, made up of the other sub-chunks below
BYTE	pad	Optional padding byte, only present if `lenChunk` is not a multiple of 2.

BMHD: Bitmap Header

The **BMHD** chunk specifies how the image is to be displayed and is usually the first chunk inside the **FORM**. It not only defines the image's height/width, but where it is drawn on the screen, how to display it in various screen resolutions and if the image is compressed. The content of this chunk is as follows:[4]

Type	Name	Description
UINT16BE	width	Image width, in pixels
UINT16BE	height	Image height, in pixels
INT16BE	xOrigin	Where on screen, in pixels, the image's top-left corner is. Value is usually 0,0 unless image is part of a larger image or not fullscreen.
INT16BE	yOrigin	
UINT8	numPlanes	Number of planes in bitmap; 1 for monochrome, 4 for 16 color, 8 for 256 color, or 0 if there is only a colormap, and no image data. (i.e., this file is just a colormap.)
UINT8	mask	1 = masked, 2 = transparent color, 3 = lasso (for MacPaint). Mask data is not considered a bit plane.
UINT8	compression	If 0 then uncompressed. If 1 then image data is RLE compressed. If 2 "Vertical RLE" from Deluxe Paint for Atari ST. Other values are theoretically possible, representing other compression methods.
UINT8	pad1	Ignore when reading, set to 0 when writing for future compatibility
UINT16BE	transClr	Transparent colour, useful only when `mask` >= 2
UINT8	xAspect	Pixel aspect, a ratio width:height; used for displaying the image on a variety of different screen resolutions for 320x200 5:6 or 10:11
UINT8	yAspect	
INT16BE	pageWidth	The size of the screen the image is to be displayed on, in pixels, usually 320×200
INT16BE	pageHeight	

BODY: Image data

The **BODY** chunk is usually the last chunk in a file,[4] and the largest.

In ILBM files the **BODY** chunk stores the actual image data as interleaved bitplanes (and optional mask) by row. The bitplanes appear first from 1 to n, followed by the mask plane. If the image is uncompressed then each line will be made up of `(width + 15) / 16` 16-bit values (i.e. one bit per pixel, rounded up to the nearest multiple of 16-bits.) If it is compressed then each line is compressed individually and is always a multiple of 16-bits long when compressed.[4]

In PBM files, the **BODY** chunk is simpler as uncompressed it is just a continuous stream of bytes containing image data.

Compression

If an image is compressed, each row of data (but not each bitplane) is compressed individually, including the mask data if present. The compression is a variety of RLE Compression using flags. It can be decoded as follows:[4]

- Loop until we have [Final length] bytes worth of data (final length calculated from image size.)
- While [Decompressed data length] < [Final length]:
  1. Read a byte [Value]
  2. If [Value] > 128, then:
     - Read the next byte and output it (257 - [Value]) times.
     - Move forward 2 bytes and return to step 1.
  3. Else if [Value] < 128, then:
     - Read and output the next [value + 1] bytes
     - Move forward [Value + 2] bytes and return to step 1.

4. Else [Value] = 128, exit the loop (stop decompressing)

For the compression routine, it's best to encode a 2 byte repeat run as a replicate run except when preceded and followed by a literal run, in which case it is best to merge the three into one literal run. Always encode >3 byte repeats as replicate runs.[4]

CAMG: Amiga mode

A **CAMG** chunk is specifically for the Commodore Amiga computer. It stores a LONG "viewport mode". This lets you specify Amiga display modes like "dual playfield" and "hold and modify". It is, not surprisingly, rare outside of Amiga games.

Type	Name	Description
UINT32BE	viewportMode	bit flags; directly interpreted by Amiga hardware

If you need to convert or display files that might contain meaningful CAMG chunks, see the 'Notes on working with ILBM files' below.

CMAP: Palette

The **CMAP** chunk contains the image's palette and consists of 3-byte RGB values for each colour used. Each byte is between 0 and 255 inclusive. The chunk is `3 × numColours` bytes long. The number of colours in the palette will be `2 ^ numBitplanes`. This chunk is optional and a default palette will be used if it is not present. It is possible to have fewer entries than expected (e.g. 7 colours for a 4-plane '16 colour' bitmap for example.) Remember that if this has an odd number of colours, as per the IFF specification the chunk will be padded by one byte to make it an even number of bytes long, but the pad byte is not included in the chunk's length field.[4]

CRNG: Colour range

The colour range chunk is 'nonstandard'. It is used by Electronic Arts' Deluxe Paint program to identify a contiguous range of colour registers or a "shade range" and colour cycling. There can be zero or more **CRNG** chunks in an ILBM file, but all should appear before the **BODY** chunk. Deluxe Paint normally writes 4 CRNG chunks in an ILBM when the user asks it to "Save Picture".[4]

Type	Name	Description
INT16BE	padding	0x0000
INT16BE	rate	Colour cycle rate. The units are such that a rate of 60 steps per second is represented as 2^{14} = 16384. Lower rates can be obtained by linear scaling: for 30 steps/second, rate = 8192.
INT16BE	flags	Flags which control the cycling of colours through the palette. If bit0 is 1, the colours should cycle, otherwise this colour register range is inactive and should have no effect. If bit1 is 0, the colours cycle upwards, i.e. each colour moves into the next index position in the colour map and the uppermost colour in the range moves down to the lowest position. If bit1 is 1, the colours cycle in the opposite direction. Only those colours between the **low** and **high** entries in the colour map should cycle.
UINT8	low	The index of the first entry in the colour map that is part of this range.
UINT8	high	The index of the last entry in the colour map that is part of this range.

CCRT: Colour cycling

Commodore's Graphicraft program uses **CCRT** for *Colour Cycling Range and Timing*. This chunk contains a CycleInfo structure. Like **CRNG** it is a nonstandard chunk.[4]

Type	Name	Description
INT16BE	direction	Cycle direction: 0=no cycling, 1=forwards, -1=backwards
UINT8	low	lowest color register selected
UINT8	high	highest color register selected
INT32BE	delaySec	Seconds between changing colors
INT32BE	delayuS	Microseconds between changing colors (added to **delaySec** to get total delay time)
INT16BE	padding	0x0000

The data is similar to a **CRNG** chunk. A program would probably only use one of these two methods of expressing colour cycle data. You could write out both if you want to communicate this information to both DeluxePaint and Graphicraft.[4]

DEST: Bitplane combining

The optional property **DEST** is a way to control how to scatter zero or more source bitplanes into a deeper destination image. Some readers may ignore DEST.[4]

Type	Name	Description
UINT8	numPlanes	Number of bitplanes in source image
UINT8	pad1	unused; use 0 for consistency
UINT16BE	planePick	How to pick planes to scatter them into the destination image
UINT16BE	planeOnOff	Default data for Plane Pick
UINT16BE	planeMask	Selects which bitplanes to store into

The low order depth number of bits in planePick, planeOnOff, and planeMask correspond one-to-one with destination bitplanes. Bit 0 with bitplane 0, etc. Any higher order bits should be ignored.[4]

"1" bits in planePick mean "put the next source bitplane into this bitplane", so the number of "1" bits should equal numPlanes. "0" bits mean "put the corresponding bit from planeOnOff into this bitplane".[4]

Bits in planeMask gate writing to the destination bitplane: "1" bits mean "write to this bitplane" while "0" bits mean "leave this bitplane alone". The normal case (with no **DEST** chunk) is equivalent to `planePick = planeMask = (2 ^ numPlanes) - 1`.[4]

Remember that color numbers are formed by pixels in the destination bitmap (depth planes deep) not in the source bitmap (numPlanes planes deep).[4]

GRAB: Hotspot

The optional **GRAB** chunk locates a "handle" or "hotspot" of the image relative to its upper left corner, e.g., when used as a mouse cursor or a "paint brush". It is optional.[4]

Type	Name	Description
INT16BE	x	X coordinate of hotspot, in pixels relative to top-left corner of the image
INT16BE	y	Y coordinate of hotspot, in pixels relative to top-left corner of the image

SPRT: Z-order

The **SPRT** chunk indicates that an image is intended to be a sprite. It should thus have a mask plane or transparent colour and shouldn't be fullscreen. How this is handled depends on the program using the image. The only data stored here is the sprite order, used by many programs to place the sprite in the foreground (a sprite of order 1 appears behind one of order 0, etc.) It is optional.[4]

Type	Name	Description
UINT16BE	order	Z-order of image (0 is closest to the foreground, larger numbers are further away/behind)

TINY: Thumbnail

The **TINY** chunk contains a small preview image for various graphics programs, including Deluxe Paint. It is compressed and is similar in format to the **BODY** chunk.

Type	Name	Description
UINT16BE	width	Thumbnail width, in pixels
UINT16BE	height	Thumbnail height, in pixels
BYTE[]	data	Pixel data, stored in exactly the same way as the **BODY** chunk. Use exactly the same algorithm, substituting the width and height from the **TINY** chunk in place of those taken from the **BMHD** chunk.

Notes for working with ILBM

Color Maps

Sometimes an ILBM file contains only a colour map and no image data. Often used to store a palette of colours that can be applied to an image separately. In this case the BODY chunk should be empty and the numPlanes field in the BMHD chunk will be 0.[4]

Deep Images

Some ILBM files contain 'true-colour' information rather than indexed colours. These so-called 'deep images' files have no CMAP chunk and usually have 24 or 32 bitplanes. The standard ordering for the bitplanes will put the least significant bit of the red component first:[4]

R0 R1 R2 R3 R4 R5 R6 R7 G0 G1 G2 G3 G4 G5 G6 G7 B0 B1 B2 B3 B4 B5 B6 B7

If there are 32 bit planes, the last 8 bit planes will be an alpha channel:

R0 R1 ... R7 G0 ... G7 B0 ... B6 B7 A0 A1 A2 A3 A4 A5 A6 A7

An image containing no colour map and only 8 bitplanes may be a greyscale image:

I0 I1 I2 I3 I4 I5 I6 I7

Extra Half-Brite

If the ILBM file contains a CAMG chunk in which bit 7 is set (i.e. 0x80 in hexadecimal). The file expects to make use of the EHB (Extra Half-Brite) mode of the Amiga chipset. The colour map will have no more than 32 entries, but the image will have 6 bitplanes. The most significant bitplane should be regarded as a flag, when unset, use the lower 5 bits as an index into the colour map as usual. When the flag is set; use the lower 5 bits as an index into the colour map, but the actual colour to be used should be half as bright, which can be achieved by shifted the RGB components of the colour one bit to the right. Alternatively, create a colour map with 64 entries, and copy the lower 32 entries into the upper half, converting them to half brightness; then use all 6 bitplanes as a colour index.[4]

PBM images cannot exist in extra half-brite mode.

Hold and Modify

If the ILBM file contains a CAMG chunk in which bit 11 is set (i.e. 0x800 in hexadecimal) the file expects to make use of the HAM (Hold-And-Modify) mode of the Amiga chipset. In HAM6 format the colour map will have up to 16 entries, but the image will have 6 (or possibly 5 bitplanes). In HAM8 format the colour map will have up to 64 entries but the image will have 8 (or possibly 7 bitplanes).[4]

The last two bitplanes (if an odd number of bitplanes assume an extra bitplane which is always 0) are control flags which indicate how to use the first 4 (or 6) bitplanes.[4]

Control Flags	Description
00	Use bitplanes 0-3 (or 0-5) as a colour map index as normal
10	Use the colour of the previous pixel but replace the Blue component with bits from bitplanes 0-3 (or 0-5)
01	Use the colour of the previous pixel but replace the Red component with bits from bitplanes 0-3 (or 0-5)
11	Use the colour of the previous pixel but replace the Green component with bits from bitplanes 0-3 (or 0-5)

If the first pixel of a scanline is a modification pixel, then modify and use the image border colour.[4]

Note that when using 4 bits to modify a colour component you should use the 4 bits in the upper 4 bits of the component AND in the lower 4 bits (to avoid reducing the overall colour gamut). When using 6 bits this is less important, but you can still put the 2 most significant bits of the modification bits into the least significant two bits of the colour component.[4]

PBM images cannot exist in hold and modify mode.

Utilities

Most utilities that work with ILBM and BBM files are rather **dated**, such as MacPaint or Deluxe Paint. IrfanView allows viewing files, is free for non commercial use, and can work under Linux. Netpbm can convert images from ILBM to its own PPM format[5] and back.[6] The Deluxe Paint-inspired GrafX2 pixel art graphics editor can load and save ILBM files. ImageMagick and GraphicsMagick can also display and convert ILBM images.

Notes

In the Commander Keen Dreams series of games, compressed standalone ILBM images are used for title screens, but the game does not read most of the ILBM chunks. This is because the images were edited in DeluxePaint, then imported directly into the game's files.

See also

- Amiga
- Interchange File Format
- IrfanView

References

1. Jerry Morrison (1985-01-14). "EA IFF 85: Standard for Interchange Format Files" (http://www.martinreddy.net/gfx/2d/IFF.txt). Electronic Arts. Retrieved 2014-03-06.
2. Jerry Morrison (1986-01-17). " "ILBM" IFF Interleaved Bitmap" (https://web.archive.org/web/20140613000343/http://home.comcast.net/~erniew/lwsdk/docs/filefmts/ilbm.html). Electronic Arts. Archived from the original (http://home.comcast.net/~erniew/lwsdk/docs/filefmts/ilbm.html) on 2014-06-13. Retrieved 2014-03-06.
3. James D. Murray, William vanRyper (April 1996). "Encyclopedia of Graphics File Formats, Second Edition" (https://archive.org/details/mac_Graphics_File_Formats_Second_Edition_1996). O'Reilly. ISBN 1-56592-161-5. Retrieved 2014-02-27.
4. Hyperion Entertainment and contributors (8 June 2012). "ILBM IFF Interleaved Bitmap" (http://wiki.amigaos.net/wiki/ILBM_IFF_Interleaved_Bitmap). Retrieved 2018-07-30.
5. Jef Poskanzer, Ingo Wilken (12 November 2014). "ilbmtoppm" (http://netpbm.sourceforge.net/doc/ilbmtoppm.html). Retrieved 2019-06-13.
6. Jef Poskanzer, Ingo Wilken (28 June 2015). "ppmtoilbm" (http://netpbm.sourceforge.net/doc/ppmtoilbm.html). Retrieved 2019-06-13.

External links

- PNG2ILBM (http://bgafc.t-hosting.hu/prgv.php?p=2) Converts PNG files to ILBM format. In theory it can convert any PNG, including alpha channeled and/or 16-bit depth per channel ones. It supports resampling, quantizing, dithering, color register preservation or override on any bitplanes from 1 to 8, including Extra-HalfBrite.
- Graphics Workshop 1.1Y (http://settlers2.net/documentation/graphics-files-lbm/) from mid-90s can convert from and to all variants of ILBM files; it supports a variety of other image file formats. It is dated but still works on even Windows 10 when running in Windows XP compatibility mode. There is also newer commercial version known as Graphics Workshop Professional (http://www.mindworkshop.com/gww.html) with much more modern UI (seeming to be mid-00s), which however is also dated by today's standards.
- Ultimate Paint (http://www.ultimatepaint.com/) can read, write and display palette color cycle animations.
- XnView's nconvert (http://www.xnview.com/en/nconvert/) is a free and **up to date** command line converter.
- Image Converter Plus (http://www.imageconverterplus.com/) is a program that will convert ILBM files into any number of formats. While the full version is not free, the demo version adds a watermark that can be removed.
- Paint Shop Pro 7.04 and other older versions of PSP can read and write ILBM, but can only read PBM files. PSP7 gets a special mention as the shareware version has a bug that allows the evaluation shutdown mechanism to be skipped by simply opening a file (i.e. modify shortcut to always open a file and you won't be bothered).