GIF

The **Graphics Interchange Format** (**GIF**; /dʒɪf/ *JIF* or /ɡɪf/ *GHIF*) is a bitmap image format that was developed by a team at the online services provider CompuServe led by American computer scientist Steve Wilhite on 15 June 1987.[1] It has since come into widespread usage on the World Wide Web due to its wide support and portability between applications and operating systems.

The format supports up to 8 bits per pixel for each image, allowing a single image to reference its own palette of up to 256 different colors chosen from the 24-bit RGB color space. It also supports animations and allows a separate palette of up to 256 colors for each frame. These palette limitations make GIF less suitable for reproducing color photographs and other images with color gradients, but well-suited for simpler images such as graphics or logos with solid areas of color. Unlike video, the GIF file format does not support audio.

GIF images are compressed using the Lempel–Ziv–Welch (LZW) lossless data compression technique to reduce the file size without degrading the visual quality. This compression technique was patented in 1985. Controversy over the licensing agreement between the software patent holder, Unisys, and CompuServe in 1994 spurred the development of the Portable Network Graphics (PNG) standard. By 2004 all the relevant patents had expired.

GIF

An animated GIF of a rotating globe

Filename extension	`.gif`
Internet media type	`image/gif`
Type code	`GIFf`
Uniform Type Identifier (UTI)	com.compuserve.gif
Magic number	`GIF87a`/`GIF89a`
Developed by	CompuServe
Initial release	15 June 1987[1]
Latest release	89a (1989[2])
Type of format	lossless bitmap image format
Website	www.w3.org/Graphics/GIF/spec-gif89a.txt (http://www.w3.org/Graphics/GIF/spec-gif89a.txt)

Contents

History
Terminology
    Pronunciation of GIF
Usage
File format
Palettes
    True color
Example GIF file
    Image coding
    Image decoding
    LZW code lengths
    Uncompressed GIF
Compression example
Interlacing

Animated GIF

Metadata

Unisys and LZW patent enforcement

Alternatives

PNG

Animation formats

Uses

See also

References

External links

History

CompuServe introduced GIF on 15 June 1987 to provide a color image format for their file downloading areas. This replaced their earlier run-length encoding format, which was black and white only. GIF became popular because it used LZW data compression. Since this was more efficient than the run-length encoding used by PCX and MacPaint, fairly large images could be downloaded reasonably quickly even with slow modems.

The original version of GIF was called **87a**.[1] In 1989, CompuServe released an enhanced version, called **89a**,[2] which added support for animation delays (multiple images in a stream were already supported in 87a), transparent background colors, and storage of application-specific metadata. The 89a specification also supports incorporating text labels as text (not embedding them in the graphical data), but as there is little control over display fonts, this feature is not widely used. The two versions can be distinguished by looking at the first six bytes of the file (the "magic number" or signature), which, when interpreted as ASCII, read "GIF87a" and "GIF89a", respectively.

CompuServe encouraged the adoption of GIF by providing downloadable conversion utilities for many computers. By December 1987, for example, an Apple IIGS user could view pictures created on an Atari ST or Commodore 64.[3] GIF was one of the first two image formats commonly used on Web sites, the other being the black-and-white XBM.[4]

In September 1995 Netscape Navigator 2.0 added the ability for animated GIFs to loop.

The feature of storing multiple images in one file, accompanied by control data, is used extensively on the Web to produce simple animations.

The optional interlacing feature, which stores image scan lines out of order in such a fashion that even a partially downloaded image was somewhat recognizable, also helped GIF's popularity,[5] as a user could abort the download if it was not what was required.

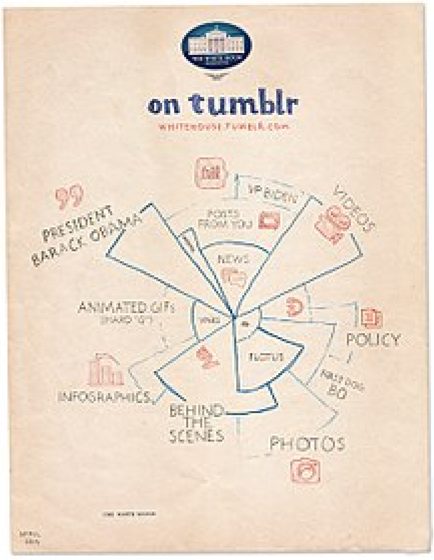
In May 2015 Facebook added support for GIF.[6][7] In January 2018 Instagram also added GIF stickers to the story mode.[8]

Terminology

As a noun, the word *GIF* is found in the newer editions of many dictionaries. In 2012, the American wing of the Oxford University Press recognized *GIF* as a verb as well, meaning "to create a GIF file", as in "GIFing was perfect medium for sharing scenes from the Summer Olympics". The press's lexicographers voted it their word of the year, saying that GIFs have evolved into "a tool with serious applications including research and journalism".[9][10]

Pronunciation of GIF

The creators of the format pronounced the word as "jif" with a soft "G" /dʒɪf/ as in "gym". Steve Wilhite says that the intended pronunciation deliberately echoes the American peanut butter brand Jif, and CompuServe employees would often say "Choosy developers choose GIF", spoofing this brand's television commercials.[11] The word is now also widely pronounced with a hard "G" /ɡɪf/ as in "gift".[12] In 2017, an informal poll on programming website Stack Overflow showed some numerical preference for hard-"G" pronunciation,[13] especially among respondents in eastern Europe, though both soft-"G" and enunciating each letter individually were found to be popular in Asia and emerging countries.[14]



A humorous image announcing the launch of a White House Tumblr suggests pronouncing GIF with the hard "G" sound.

The *American Heritage Dictionary*[15] cites both, indicating "jif" as the primary pronunciation, while *Cambridge Dictionary of American English*[16] offers only the hard-"G" pronunciation. *Merriam-Webster's Collegiate Dictionary*[17] and the *OED* cite both pronunciations, but place "gif" in the default position ("\ˈgif, ˈjif\").[18] The *New Oxford American Dictionary* gave only "jif" in its 2nd edition[19] but updated it to "jif, gif" in its 3rd edition.[20]

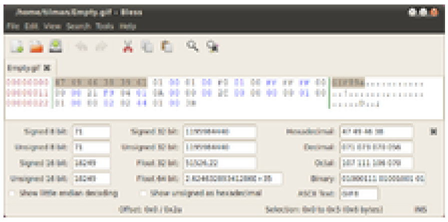
The disagreement over the pronunciation led to heated Internet debate. On the occasion of receiving a lifetime achievement award at the 2013 Webby Award ceremony, Wilhite rejected the hard-"G" pronunciation,[12][21][22] and his speech led to 17,000 posts on Twitter and 50 news articles.[23] The White House[12] and TV program *Jeopardy!* also entered the debate during 2013.[22]

In February 2020, The J.M. Smucker Company, the owners of the Jif peanut butter brand, partnered with animated image database and search engine Giphy to release a limited-edition "Jif vs. GIF" (hashtagged as #JIFvsGIF) jar of Jif peanut butter that has a label humorously declaring the soft-"G" pronunciation to exclusively refer to the peanut butter, and GIF to be exclusively pronounced with the hard-"G" pronunciation.[24]

Usage

- GIFs are suitable for sharp-edged line art with a limited number of colors, such as logos. This takes advantage of the format's lossless compression, which favors flat areas of uniform color with well defined edges.[25]
- GIFs may be used to store low-color sprite data for games.[26]
- GIFs can be used for small animations and low-resolution video clips.[26]
- GIFs can be used as a reaction when messaging online, used to convey emotion and feelings, alternative to using words
- Popular on social media platforms such as Tumblr, Facebook and Twitter.

File format



File:Empty.gif in a hex editor

Conceptually, a GIF file describes a fixed-sized graphical area (the "logical screen") populated with zero or more "images". Many GIF files have a single image that fills the entire logical screen. Others divide the logical screen into separate sub-images. The images may also function as animation frames in an animated GIF file, but again these need not fill the entire logical screen.

GIF files start with a fixed-length header ("GIF87a" or "GIF89a") giving the version, followed by a fixed-length Logical Screen Descriptor giving the pixel dimensions and other characteristics of the logical screen. The screen descriptor may also specify the presence and size of a Global Color Table, which follows next if present.

```
00000000  47 49 46 38 39 61 01 00  01 00 80 00 00 00 00 00  |GIF89a..........|
00000010  ff ff ff 21 f9 04 01 00  00 00 00 2c 00 00 00 00  |...!.......,....|
00000020  01 00 01 00 00 02 01 44  00 3b                    |.......D.;|
0000002a
```

Thereafter, the file is divided into segments, each introduced by a 1-byte sentinel:

- An image (introduced by 0x2C, an ASCII comma `','`)
- An extension block (introduced by 0x21, an ASCII exclamation point `'!'`)
- The trailer (a single byte of value 0x3B, an ASCII semicolon `';'`), which should be the last byte of the file.

An image starts with a fixed-length Image Descriptor, which may specify the presence and size of a Local Color Table (which follows next if present). The image data follows: one byte giving the bit width of the unencoded symbols (which must be at least 2 bits wide, even for bi-color images), followed by a linked list of sub-blocks containing the LZW-encoded data.

Extension blocks (blocks that "extend" the 87a definition via a mechanism already defined in the 87a spec) consist of the sentinel, an additional byte specifying the type of extension, and a linked list of sub-blocks with the extension data. Extension blocks that modify an image (like the Graphic Control Extension that specifies the optional animation delay time and optional transparent background color) must immediately precede the segment with the image they refer to.

The linked lists used by the image data and the extension blocks consist of series of sub-blocks, each sub-block beginning with a byte giving the number of subsequent data bytes in the sub-block (1 to 255). The series of sub-blocks is terminated by an empty sub-block (a 0 byte).

This structure allows the file to be parsed even if not all parts are understood. A GIF marked 87a may contain extension blocks; the intent is that a decoder can read and display the file without the features covered in extensions it does not understand.

The full detail of the file format is covered in the GIF specification.[2]

Palettes

GIF is palette-based: the colors used in an image (a frame) in the file have their RGB values defined in a palette table that can hold up to 256 entries, and the data for the image refer to the colors by their indices (0–255) in the palette table. The color definitions in the palette can be drawn from a color space of millions of shades (2^{24} shades, 8 bits for each primary), but the maximum number of colors a frame can use is 256. This limitation seemed reasonable when GIF was developed because few people could afford the hardware to display more colors simultaneously. Simple graphics, line drawings, cartoons, and grey-scale photographs typically need fewer than 256 colors.

An example of a GIF image saved with a web-safe palette and dithered using the Floyd–Steinberg method. Due to the reduced number of colors in the image, there are display issues.

Each frame can designate one index as a "transparent background color": any pixel assigned this index takes on the color of the pixel in the same position from the background, which may have been determined by a previous frame of animation.

Many techniques, collectively called dithering, have been developed to approximate a wider range of colors with a small color palette by using pixels of two or more colors to approximate in-between colors. These techniques sacrifice spatial resolution to approximate deeper color resolution. While not part of the GIF specification, dithering can be used in images subsequently encoded as GIF images. This is often not an ideal solution for GIF images, both because the loss of spatial resolution typically makes an image look fuzzy on the screen, and because the dithering patterns often interfere with the compressibility of the image data, working against GIF's main purpose.

In the early days of graphical web browsers, graphics cards with 8-bit buffers (allowing only 256 colors) were common and it was fairly common to make GIF images using the websafe palette. This ensured predictable display, but severely limited the choice of colors. When 24-bit color became the norm palettes could instead be populated with the optimum colors for individual images.

A small color table may suffice for small images, and keeping the color table small allows the file to be downloaded faster. Both the 87a and 89a specifications allow color tables of 2^n colors for any n from 1 through 8. Most graphics applications will read and display GIF images with any of these table sizes; but some do not support all sizes when *creating* images. Tables of 2, 16, and 256 colors are widely supported.

True color

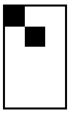
Although GIF is almost never used for true color images, it is possible to do so.[27][28] A GIF image can include multiple image blocks, each of which can have its own 256-color palette, and the blocks can be tiled to create a complete image. Alternatively, the GIF89a specification introduced the idea of a "transparent" color where each image block can include its own palette of 255 visible colors plus one transparent color. A complete image can be created by layering image blocks with the visible portion of each layer showing through the transparent portions of the layers above.

To render a full-color image as a GIF, the original image must be broken down into smaller regions having no more than 255 or 256 different colors. Each of these regions is then stored as a separate image block with its own local palette and when the image blocks are displayed together (either by tiling or by layering partially transparent image blocks) the complete, full-color image appears. For example, breaking an image into tiles of 16 by 16 pixels (256 pixels in total) ensures that no tile has more than the local palette limit of 256 colors, although larger tiles may be used and similar colors merged resulting in some loss of color information.[27]

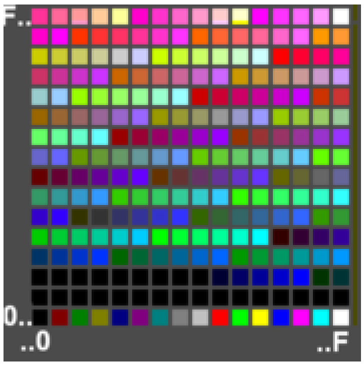
Since each image block can have its own local color table, a GIF file having many image blocks can be very large, limiting the usefulness of full-color GIFs.[28] Additionally, not all GIF rendering programs handle tiled or layered images correctly. Many rendering programs interpret tiles or layers as animation frames and display them in sequence as an endless animation[27] with most web browsers automatically displaying the frames with a delay time of 0.1 seconds or more.[29][30]

An animated GIF illustrating a technique for displaying more than the typical limit of 256 colors

Example GIF file



Sample image (enlarged), actual size 3 pixels wide by 5 high



Bytes D_h to $30C_h$ in the example define a palette of 256 colors.

Microsoft Paint saves a small black-and-white image as the following GIF file. Paint does not make optimal use of GIF; due to the unnecessarily large color table (storing a full 256 colors instead of the used 2) and symbol width, this GIF file is not an efficient representation of the 15-pixel image (illustrated enlarged above).

Although the Graphic Control Extension block declares color index 16 (hexadecimal 10) to be transparent, that index is not used in the image. The only color indexes appearing in the image data are decimal 40 and 255, which the Global Color Table maps to black and white, respectively.

Note that the hex numbers in the following tables are in little-endian byte order, as the format specification prescribes.

```
byte#  hexadecimal  text or
(hex)               value        Meaning
0:     47 49 46
       38 39 61     GIF89a       Header
                                 Logical Screen Descriptor
6:     03 00        3             - logical screen width in pixels
8:     05 00        5             - logical screen height in pixels
A:     F7                         - GCT follows for 256 colors with resolution 3×8 bits/primary;
                                    the lowest 3 bits represent the bit depth minus 1, the
highest true bit means that the GCT is present
B:     00           0             - background color #0
C:     00                         - default pixel aspect ratio
                   R    G    B   Global Color Table
D:     00 00 00    0    0    0    - color #0 black
10:    80 00 00  128    0    0    - color #1
 :                                       :
85:    00 00 00    0    0    0    - color #40 black
 :                                       :
30A:   FF FF FF  255  255  255    - color #255 white
30D:   21 F9                     Graphic Control Extension (comment fields precede this in most
files)
30F:   04            4            - 4 bytes of GCE data follow
310:   01                         - there is a transparent background color (bit field; the
lowest bit signifies transparency)
311:   00 00                      - delay for animation in hundredths of a second: not used
313:   10           16            - color #16 is transparent
314:   00                         - end of GCE block
```

```
315:   2C                         Image Descriptor
316:   00 00 00 00 (0,0)           - NW corner position of image in logical screen
31A:   03 00 05 00 (3,5)           - image width and height in pixels
31E:   00                          - no local color table
31F:   08           8             Start of image - LZW minimum code size
320:   0B          11              - 11 bytes of LZW encoded image data follow
321:   00 51 FC 1B 28 70 A0 C1 83 01 01
32C:   00                          - end of image data
32D:   3B                         GIF file terminator
```

Image coding

The image pixel data, scanned horizontally from top left, are converted by LZW encoding to codes that are then mapped into bytes for storing in the file. The pixel codes typically don't match the 8-bit size of the bytes, so the codes are packed into bytes by a "little-Endian" scheme: the least significant bit of the first code is stored in the least significant bit of the first byte, higher order bits of the code into higher order bits of the byte, spilling over into the low order bits of the next byte as necessary. Each subsequent code is stored starting at the least significant bit not already used.

This byte stream is stored in the file as a series of "sub-blocks". Each sub-block has a maximum length 255 bytes and is prefixed with a byte indicating the number of data bytes in the sub-block. The series of sub-blocks is terminated by an empty sub-block (a single 0 byte, indicating a sub-block with 0 data bytes).

For the sample image above the reversible mapping between 9-bit codes and bytes is shown below.

```
9-bit code  Binary  Bytes
   (hex)            (hex)
            00000000|    00
100
            0101000|1    51
028
            111111|00    FC
0FF
            00011|011    1B
103
            0010|1000    28
102
            011|10000    70
103
            10|100000    A0
106
            1|1000001    C1
107
             |10000011   83
101
             00000001    01
            0000000|1    01
```

A slight compression is evident: pixel colors defined initially by 15 bytes are exactly represented by 12 code bytes including control codes. The encoding process that produces the 9-bit codes is shown below. A local string accumulates pixel color numbers from the palette, with no output action as long as the local string can be found in a code table. There is special treatment of the first two pixels that arrive before the table grows from its initial size by additions of strings. After each output code, the local string is initialized to the latest pixel color (that could not be included in the output code).

```
         Table          9-bit
   string --> code      code     Action
        #0 | 000h                Initialize root table of 9-bit codes
  palette  |  :
   colors  |  :
      #255 | 0FFh
       clr | 100h
```

```
                           end | 101h
                               |           100h     Clear
Pixel          Local           |
color Palette  string          |
BLACK  #40     28              |           028h     1st pixel always to output
WHITE  #255    FF              |                    String found in table
                  28 FF        | 102h               Always add 1st string to table
               FF              |                    Initialize local string
WHITE  #255    FF FF           |                    String not found in table
                               |           0FFh     - output code for previous string
                  FF FF        | 103h               - add latest string to table
               FF              |                    - initialize local string
WHITE  #255    FF FF           |                    String found in table
BLACK  #40     FF FF 28        |                    String not found in table
                               |           103h     - output code for previous string
                  FF FF 28     | 104h               - add latest string to table
               28              |                    - initialize local string
WHITE  #255    28 FF           |                    String found in table
WHITE  #255    28 FF FF        |                    String not found in table
                               |           102h     - output code for previous string
                  28 FF FF     | 105h               - add latest string to table
               FF              |                    - initialize local string
WHITE  #255    FF FF           |                    String found in table
WHITE  #255    FF FF FF        |                    String not found in table
                               |           103h     - output code for previous string
                  FF FF FF     | 106h               - add latest string to table
               FF              |                    - initialize local string
WHITE  #255    FF FF           |                    String found in table
WHITE  #255    FF FF FF        |                    String found in table
WHITE  #255    FF FF FF FF     |                    String not found in table
                               |           106h     - output code for previous string
                  FF FF FF FF| 107h                 - add latest string to table
               FF              |                    - initialize local string
WHITE  #255    FF FF           |                    String found in table
WHITE  #255    FF FF FF        |                    String found in table
WHITE  #255    FF FF FF FF     |                    String found in table
                                                    No more pixels
                                           107h     - output code for last string
                                           101h     End
```

For clarity the table is shown above as being built of strings of increasing length. That scheme can function but the table consumes an unpredictable amount of memory. Memory can be saved in practice by noting that each new string to be stored consists of a previously stored string augmented by one character. It is economical to store at each address only two words: an existing address and one character.

The LZW algorithm requires a search of the table for each pixel. A linear search through up to 4096 addresses would make the coding slow. In practice the codes can be stored in order of numerical value; this allows each search to be done by a SAR (Successive Approximation Register, as used in some ADCs), with only 12 magnitude comparisons. For this efficiency an extra table is needed to convert between codes and actual memory addresses; the extra table upkeeping is needed only when a new code is stored which happens at much less than pixel rate.

Image decoding

Decoding begins by mapping the stored bytes back to 9-bit codes. These are decoded to recover the pixel colors as shown below. A table identical to the one used in the encoder is built by adding strings by this rule:

Is incoming code found in table?

Yes	add string for local code followed by first byte of string for incoming code
No	add string for local code followed by copy of its own first byte

```
      shift
9-bit ----> Local      Table                Pixel
code        code   code --> string   Palette color  Action
100h               000h  | #0                        Initialize root table of 9-bit codes
                    :    | palette
```

```
                    :    | colors
                   0FFh  | #255
                   100h  | clr
                   101h  | end
028h                     |             #40   BLACK  Decode 1st pixel
0FFh        028h         |                           Incoming code found in table
                         |             #255  WHITE   - output string from table
                   102h  | 28 FF                     - add to table
103h        0FFh         |                           Incoming code not found in table
                   103h  | FF FF                     - add to table
                         |                           - output string from table
                         |             #255  WHITE
                         |             #255  WHITE
102h        103h         |                           Incoming code found in table
                         |                           - output string from table
                         |             #40   BLACK
                         |             #255  WHITE
                   104h  | FF FF 28                  - add to table
103h        102h         |                           Incoming code found in table
                         |                           - output string from table
                         |             #255  WHITE
                         |             #255  WHITE
                   105h  | 28 FF FF                  - add to table
106h        103h         |                           Incoming code not found in table
                   106h  | FF FF FF                  - add to table
                         |                           - output string from table
                         |             #255  WHITE
                         |             #255  WHITE
                         |             #255  WHITE
107h        106h         |                           Incoming code not found in table
                   107h  | FF FF FF FF               - add to table
                         |                           - output string from table
                         |             #255  WHITE
                         |             #255  WHITE
                         |             #255  WHITE
                         |             #255  WHITE
101h                     |                           End
```

LZW code lengths

Shorter code lengths can be used for palettes smaller than the 256 colors in the example. If the palette is only 64 colors (so color indexes are 6 bits wide), the symbols can range from 0 to 63, and the symbol width can be taken to be 6 bits, with codes starting at 7 bits. In fact, the symbol width need not match the palette size: as long as the values decoded are always less than the number of colors in the palette, the symbols can be any width from 2 to 8, and the palette size any power of 2 from 2 to 256. For example, if only the first four colors (values 0 to 3) of the palette are used, the symbols can be taken to be 2 bits wide with codes starting at 3 bits.

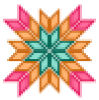
Conversely, the symbol width could be set at 8, even if only values 0 and 1 are used; these data would only require a two-color table. Although there would be no point in encoding the file that way, something similar typically happens for bi-color images: the minimum symbol width is 2, even if only values 0 and 1 are used.

The code table initially contains codes that are one bit longer than the symbol size in order to accommodate the two special codes *clr* and *end* and codes for strings that are added during the process. When the table is full the code length increases to give space for more strings, up to a maximum code 4095 = FFF(hex). As the decoder builds its table it tracks these increases in code length and it is able to unpack incoming bytes accordingly.

Uncompressed GIF

The GIF encoding process can be modified to create a file without LZW compression that is still viewable as a GIF image. This technique was introduced originally as a way to avoid patent infringement. Uncompressed GIF can also be a useful intermediate format for a graphics programmer because individual pixels are accessible for reading or painting. An uncompressed GIF file can be converted to an ordinary GIF file simply by passing it through an image editor.

The modified encoding method ignores building the LZW table and emits only the root palette codes and the codes for CLEAR and STOP. This yields a simpler encoding (a 1-to-1 correspondence between code values and palette codes) but sacrifices all of the compression: each pixel in the image generates an output code indicating its color index. When processing an uncompressed GIF, a standard GIF decoder will not be prevented from writing strings to its dictionary table, but the code width must never increase since that triggers a different packing of bits to bytes.



A 46×46 uncompressed GIF with 7-bit symbols (128 colors, 8-bit codes). Click on the image for an explanation of the code.

If the symbol width is n, the codes of width $n+1$ fall naturally into two blocks: the lower block of 2^n codes for coding single symbols, and the upper block of 2^n codes that will be used by the decoder for sequences of length greater than one. Of that upper block, the first two codes are already taken: 2^n for CLEAR and $2^n + 1$ for STOP. The decoder must also be prevented from using the last code in the upper block, $2^{n+1} - 1$, because when the decoder fills that slot, it will increase the code width. Thus in the upper block there are $2^n - 3$ codes available to the decoder that won't trigger an increase in code width. Because the decoder is always one step behind in maintaining the table, it does not generate a table entry upon receiving the first code from the encoder, but will generate one for each succeeding code. Thus the encoder can generate $2^n - 2$ codes without triggering an increase in code width. Therefore, the encoder must emit extra CLEAR codes at intervals of $2^n - 2$ codes or less to make the decoder reset the coding dictionary. The GIF standard allows such extra CLEAR codes to be inserted in the image data at any time. The composite data stream is partitioned into sub-blocks that each carry from 1 to 255 bytes.

For the sample 3×5 image above, the following 9-bit codes represent "clear" (100) followed by image pixels in scan order and "stop" (101).

```
9-bit codes: 100 028 0FF 0FF 0FF 028 0FF 0FF 0FF 0FF 0FF 0FF 0FF 0FF 0FF 0FF 101
```

After the above codes are mapped to bytes, the uncompressed file differs from the compressed file thus:

```
 :
320: 14             20             20 bytes uncompressed image data follow
321: 00 51 FC FB F7 0F C5 BF 7F FF FE FD FB F7 EF DF BF 7F 01 01
335: 00                            - end
 :
```

Compression example

The trivial example of a large image of solid color demonstrates the variable-length LZW compression used in GIF files.

Sample compression of a GIF file

Code			**Pixels**		**Notes**
No. N_i	**Value** $N_i + 256$	**Length (bits)**	**This code** N_i	**Accumulated** $\frac{N_i(N_i+1)}{2}$	Relations using N_i only apply to same-color pixels until coding table is full.
0	100h	9			Clear code table
1	FFh		1	1	Top left pixel color chosen as the highest index of a 256-color palette
2	102h		2	3	
3 ⋮ 255	103h ⋮ 1FFh		3 ⋮ 255	6 ⋮ 32640	Last 9-bit code
256 ⋮ 767	200h ⋮ 3FFh	10	256 ⋮ 767	32896 ⋮ 294528	Last 10-bit code
768 ⋮ 1791	400h ⋮ 7FFh	11	768 ⋮ 1791	295296 ⋮ 1604736	Last 11-bit code
1792 ⋮ 3839	800h ⋮ FFFh	12	1792 ⋮ 3839	1606528 ⋮ 7370880	Code table full
⋮	FFFh		3839	The maximum code may repeat for more same-color pixels. Overall data compression asymptotically approaches $3839 \times \frac{8}{12} = 2559\frac{1}{3}$	
	101h				End of image data

The code values shown are packed into bytes which are then packed into blocks of up to 255 bytes. A block of image data begins with a byte that declares the number of bytes to follow. The last block of data for an image is marked by a zero block-length byte.

Interlacing

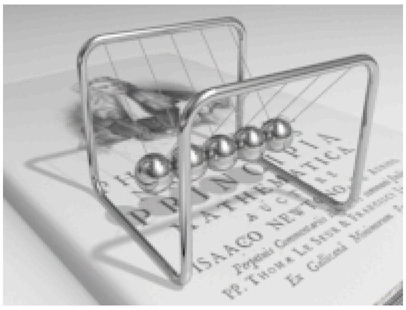
The GIF Specification allows each image within the logical screen of a GIF file to specify that it is interlaced; i.e., that the order of the raster lines in its data block is not sequential. This allows a partial display of the image that can be recognized before the full image is painted.

An interlaced image is divided from top to bottom into strips 8 pixels high, and the rows of the image are presented in the following order:

- Pass 1: Line 0 (the top-most line) from each strip.
- Pass 2: Line 4 from each strip.
- Pass 3: Lines 2 and 6 from each strip.
- Pass 4: Lines 1, 3, 5, and 7 from each strip.

The pixels within each line are not interlaced, but presented consecutively from left to right. As with non-interlaced images, there is no break between the data for one line and the data for the next. The indicator that an image is interlaced is a bit set in the corresponding Image Descriptor block.

Animated GIF



GIF can be used to display animation, as in this image of Newton's Cradle.

Although GIF was not designed as an animation medium, its ability to store multiple images in one file naturally suggested using the format to store the frames of an animation sequence. To facilitate *displaying* animations, the GIF89a spec added the Graphic Control Extension (GCE), which allows the images (frames) in the file to be painted with time delays, forming a video clip. Each frame in an animation GIF is introduced by its own GCE specifying the time delay to wait after the frame is drawn. Global information at the start of the file applies by default to all frames. The data is stream-oriented, so the file offset of the start of each GCE depends on the length of preceding data. Within each frame the LZW-coded image data is arranged in sub-blocks of up to 255 bytes; the size of each sub-block is declared by the byte that precedes it.

A GIF animation made of two photos, one morphing into the other

By default, an animation displays the sequence of frames only once, stopping when the last frame is displayed. To enable an animation to loop, Netscape in the 1990s used the Application Extension block (intended to allow vendors to add application-specific information to the GIF file) to implement the Netscape Application Block (NAB).[31] This block, placed immediately before the sequence of animation frames, specifies the number of times the sequence of frames should be played (1 to 65535 times) or that it should repeat continuously (zero indicates loop forever). Support for these repeating animations first appeared in Netscape Navigator version 2.0, and then spread to other browsers.[32] Most browsers now recognize and support NAB, though it is not strictly part of the GIF89a specification.

The following example shows the structure of the animation file *Rotating earth (large).gif* shown (as a thumbnail) in the article's infobox.

```
byte#   hexadecimal  text or
(hex)                value      Meaning
0:      47 49 46
        38 39 61     GIF89a     Header (https://web.archive.org/web/20160304075538/http://qalle.ne
t/gif89a.php#header)
                                Logical Screen Descriptor (https://web.archive.org/web/20160304075
538/http://qalle.net/gif89a.php#logicalscreendescriptor)
6:      90 01        400         - width in pixels
8:      90 01        400         - height in pixels
A:      F7                       - GCT follows for 256 colors with resolution 3 x 8bits/primary
B:      00           0           - background color #0
C:      00                       - default pixel aspect ratio
D:                              Global Color Table (https://web.archive.org/web/20160304075538/htt
p://qalle.net/gif89a.php#globalcolortable)
:
30D:    21 FF                   Application Extension (https://web.archive.org/web/20160304075538/
http://qalle.net/gif89a.php#applicationextension) block
30F:    0B           11          - eleven bytes of data follow
310:    4E 45 54
        53 43 41
        50 45        NETSCAPE    - 8-character application name
        32 2E 30     2.0         - application "authentication code"
31B:    03           3           - three more bytes of data
31C:    01           1           - index of the current data sub-block (always 1 for the NETSCAPE
block)
31D:    FF FF        65535       - unsigned number of repetitions
31F:    00                       - end of App Extension block
320:    21 F9                   Graphic Control Extension (https://web.archive.org/web/20160304075
538/http://qalle.net/gif89a.php#graphiccontrolextension) for frame #1
322:    04           4           - four bytes in the current block
323:    04           000.....    - reserved; 5 lower bits are bit field
                     ...001..    - disposal method 1: do not dispose
                     ......0.    - no user input
                     .......0    - transparent color is not given
```

```
324:   09 00                    - 0.09 sec delay before painting next frame
326:   FF                       - transparent color index (unused in this frame)
327:   00                       - end of GCE block
328:   2C                      Image Descriptor (https://web.archive.org/web/20160304075538/htt
p://qalle.net/gif89a.php#imagedescriptor) of frame #1
329:   00 00 00 00  (0,0)       - NW corner of frame at 0, 0
32D:   90 01 90 01  (400,400)   - Frame width and height: 400×400
331:   00                       - no local color table; no interlace
332:   08             8        LZW min code size; Image Data (https://web.archive.org/web/2016030
4075538/http://qalle.net/gif89a.php#tablebasedimagedata) of frame #1 beginning
333:   FF             255       - 255 bytes of LZW encoded image data follow
334:                  data
433:   FF             255       - 255 bytes of LZW encoded image data follow
                      data
                       :
92C0:  00                       - end of LZW data for this frame
92C1:  21 F9                   Graphic Control Extension for frame #2
 :                                                              :
EDABD: 21 F9                   Graphic Control Extension for frame #44
 :
F48F5: 3B                      File terminator (https://web.archive.org/web/20160304075538/htt
p://qalle.net/gif89a.php#trailer)
```

The animation delay for each frame is specified in the GCE in hundredths of a second. Some economy of data is possible where a frame need only rewrite a portion of the pixels of the display, because the Image Descriptor can define a smaller rectangle to be rescanned instead of the whole image. Browsers or other displays that do not support animated GIFs typically show only the first frame.

The size and color quality of animated GIF files can vary significantly depending on the application used to create them. Strategies for minimizing file size include using a common global color table for all frames (rather than a complete local color table for each frame) and minimizing the number of pixels covered in successive frames (so that only the pixels that change from one frame to the next are included in the latter frame). Simply packing a series of independent frame images into a composite animation tends to yield large file sizes.

Internet Explorer slows down GIFs if the frame-rate is 20 frames per second or higher and Microsoft reports that Google Chrome and Safari also slow down some GIF animations.[33]

Starting in early 1995, the University of Ulm used animated GIF as live video streaming format to show a controllable model railroad.

Metadata

Metadata can be stored in GIF files as a comment block, a plain text block, or an application-specific application extension block. Several graphics editors use unofficial application extension blocks to include the data used to generate the image, so that it can be recovered for further editing.

All of these methods technically require the metadata to be broken into sub-blocks so that applications can navigate the metadata block without knowing its internal structure.

The Extensible Metadata Platform (XMP) metadata standard introduced an unofficial but now widespread "XMP Data" application extension block for including XMP data in GIF files.[34] Since the XMP data is encoded using UTF-8 without NUL characters, there are no 0 bytes in the data. Rather than break the data into formal sub-blocks, the extension block terminates with a "magic trailer" that routes any application treating the data as sub-blocks to a final 0 byte that terminates the sub-block chain.

Unisys and LZW patent enforcement

In 1977 and 1978, Jacob Ziv and Abraham Lempel published a pair of papers on a new class of lossless data-compression algorithms, now collectively referred to as LZ77 and LZ78. In 1983, Terry Welch developed a fast variant of LZ78 which was named Lempel–Ziv–Welch (LZW).[35][36]

Welch filed a patent application for the LZW method in June 1983. The resulting patent, US 4558302 (https://worldwide.espacenet.com/textdoc?DB=EPODOC&IDX=US4558302), granted in December 1985, was assigned to Sperry Corporation who subsequently merged with Burroughs Corporation in 1986 and formed Unisys.[35] Further patents were obtained in the United Kingdom, France, Germany, Italy, Japan and Canada.

In addition to the above patents, Welch's 1983 patent also includes citations to several other patents that influenced it, including two 1980 Japanese patents (JP9343880A (https://patents.google.com/patent/JPS5719857A/en) and JP17790880A (https://patents.google.com/patent/JPS57101937A/en)) from NEC's Jun Kanatsu, U.S. Patent 4,021,782 (https://www.google.com/patents/US4021782) (1974) from John S. Hoerning, U.S. Patent 4,366,551 (https://www.google.com/patents/US4366551) (1977) from Klaus E. Holtz, and a 1981 Dutch patent (DE19813118676 (https://patents.google.com/patent/DE3118676C2/en)) from Karl Eckhart Heinz.[37]

In June 1984, an article by Welch was published in the IEEE magazine which publicly described the LZW technique for the first time.[38] LZW became a popular data compression technique and, when the patent was granted, Unisys entered into licensing agreements with over a hundred companies.[35][39]

The popularity of LZW led CompuServe to choose it as the compression technique for their version of GIF, developed in 1987. At the time, CompuServe was not aware of the patent.[35] Unisys became aware that the version of GIF used the LZW compression technique and entered into licensing negotiations with CompuServe in January 1993. The subsequent agreement was announced on 24 December 1994.[36] Unisys stated that they expected all major commercial on-line information services companies employing the LZW patent to license the technology from Unisys at a reasonable rate, but that they would not require licensing, or fees to be paid, for non-commercial, non-profit GIF-based applications, including those for use on the on-line services.[39]

Following this announcement, there was widespread condemnation of CompuServe and Unisys, and many software developers threatened to stop using GIF. The PNG format (see below) was developed in 1995 as an intended replacement.[35][36][38] However, obtaining support from the makers of Web browsers and other software for the PNG format proved difficult and it was not possible to replace GIF, although PNG has gradually increased in popularity.[35] Therefore, GIF variations without LZW compression were developed. For instance the libungif library, based on Eric S. Raymond's giflib, allows creation of GIFs that followed the data format but avoided the compression features, thus avoiding use of the Unisys LZW patent.[40] A 2001 *Dr. Dobb's* article described another alternative to LZW compression, based on square roots.[41]

In August 1999, Unisys changed the details of their licensing practice, announcing the option for owners of certain non-commercial and private websites to obtain licenses on payment of a one-time license fee of $5000 or $7500.[42] Such licenses were not required for website owners or other GIF users who had used licensed software to generate GIFs. Nevertheless, Unisys was subjected to thousands of online attacks and abusive emails from users believing that they were going to be charged $5000 or sued for using GIFs on their websites.[43] Despite giving free licenses to hundreds of non-profit organizations, schools and governments, Unisys was completely unable to generate any good publicity and continued to be condemned by individuals and organizations such as the League for Programming Freedom who started the "Burn All GIFs" campaign in 1999.[44][45]

The United States LZW patent expired on 20 June 2003.[46] The counterpart patents in the United Kingdom, France, Germany and Italy expired on 18 June 2004, the Japanese patents expired on 20 June 2004, and the Canadian patent expired on 7 July 2004.[46] Consequently, while Unisys has further patents and patent applications relating to improvements to the LZW technique,[46] GIF may now be used freely.[47]

Alternatives

PNG

Portable Network Graphics (PNG) was designed as a replacement for GIF in order to avoid infringement of Unisys' patent on the LZW compression technique.[35] PNG offers better compression and more features than GIF,[48] animation being the only significant exception. PNG is more suitable than GIF in instances where true-color imaging and alpha transparency are required.

Although support for PNG format came slowly, new web browsers generally support PNG. Older versions of Internet Explorer do not support all features of PNG. Versions 6 and earlier do not support alpha channel transparency without using Microsoft-specific HTML extensions.[49] Gamma correction of PNG images was not supported before version 8, and the display of these images in earlier versions may have the wrong tint.[50]

For identical 8-bit (or lower) image data, PNG files are typically smaller than the equivalent GIFs, due to the more efficient compression techniques used in PNG encoding.[51] Complete support for GIF is complicated chiefly by the complex canvas structure it allows, though this is what enables the compact animation features.

Animation formats

Videos resolve many issues that GIFs present through common usage on the web. They include drastically smaller file sizes, the ability to surpass the 8-bit color restriction, and better frame-handling and compression through codecs. Virtually universal support for the GIF format in web browsers and a lack of official support for video in the HTML standard caused GIF to rise to prominence for the purpose of displaying short video-like files on the web.

MNG ("Multiple-image Network Graphics") was originally developed as a PNG-based solution for animations. MNG reached version 1.0 in 2001, but few applications support it.

In 2006, an extension to the PNG format called APNG ("Animated Portable Network Graphics") was proposed as alternative to the MNG format by Mozilla. APNG is supported by most browsers as of 2019.[52] APNG provide the ability to animate PNG files, while retaining backwards compatibility in decoders that cannot understand the animation chunk (unlike MNG). Older decoders will simply render the first frame of the animation. The PNG group officially rejected APNG as an official extension on 20 April 2007.[53] There have been several subsequent proposals for a simple animated graphics format based on PNG using several different approaches.[54] Nevertheless, Animated Portable Network Graphics is still under development by Mozilla and is supported in Firefox 3[55][56] while MNG support was dropped.[57][58] APNG is currently supported by all major web browsers including Chrome since version 59.0 and Opera and Firefox and Edge.

Embedded Adobe Flash objects and MPEGs are used on some websites to display simple video, but require the use of an additional browser plugin. WebM and WebP are in development and are supported by some web browsers.[59] Other options for web animation include serving individual frames using AJAX, or animating SVG images using JavaScript or SMIL ("Synchronized Multimedia Integration Language").

With the introduction of widespread support of the HTML5 video (`<video>`) tag in most web browsers, some websites use a looped version of the video tag generated by JavaScript functions. This gives the appearance of a GIF, but with the size and speed advantages of compressed video. Notable examples are

Gfycat and Imgur and their GIFV metaformat, which is really a video tag playing a looped MP4 or WebM compressed video.[60]

High Efficiency Image File Format (HEIF) is an image file format, finalized in 2015, which uses a discrete cosine transform (DCT) lossy compression algorithm based on the HEVC video format, and related to the JPEG image format. In contrast to JPEG, HEIF supports animation.[61] Compared to the GIF format, which lacks DCT compression, HEIF allows significantly more efficient compression. HEIF stores more information and produces higher-quality animated images at a small fraction of an equivalent GIF's size.[62]

VP9 only supports alpha compositing with 4:2:0 chroma subsampling[63] in the YUVA420 pixel format, which may be unsuitable for GIFs that combine transparency with rasterised vector graphics with fine color details.

Uses

In April 2014, 4chan added support for silent WebM videos that are under 3 MB in size and 2 min in length,[64][65] and in October 2014, Imgur started converting any GIF files uploaded to the site to video and giving the link to the HTML player the appearance of an actual file with a `.gifv` extension.[66][67]

In January 2016, Telegram started re-encoding all GIFs to MPEG4 videos that "require up to 95% less disk space for the same image quality."[68]

See also

- Cinemagraph, a partially animated photograph often in GIF
- Comparison of graphics file formats
- Comparison of layout engines (graphics)
- GIF art, a form of digital art associated with GIF
- GNU plotutils (supports pseudo-GIF, which uses run-length encoding rather than LZW)
- Microsoft GIF Animator, historic program to create simple animated GIFs
- Software patent

References

1. "Graphics Interchange Format, Version 87a" (http://www.w3.org/Graphics/GIF/spec-gif87.txt). W3C. 15 June 1987. Retrieved 13 October 2012.
2. "Graphics Interchange Format, Version 89a" (http://www.w3.org/Graphics/GIF/spec-gif89a.txt). W3C. 31 July 1990. Retrieved 6 March 2009.
3. "Online Art" (https://archive.org/stream/COMPUTEs_Apple_Applications_Vol._5_No._2_Issue_6_1987-12_COMPUTE_Publications_US#page/n11/mode/2up). *Compute!'s Apple Applications*. December 1987. p. 10. Retrieved 14 September 2016.
4. Holdener III, Anthony (2008). *Ajax: The Definitive Guide: Interactive Applications for the Web*. O'Reilly Media. ISBN 978-0596528386.
5. Furht, Borko (2008). *Encyclopedia of Multimedia*. Springer. ISBN 978-0387747248.
6. McHugh, Molly (29 May 2015). "You Can Finally, Actually, Really, Truly Post GIFs on Facebook" (https://www.wired.com/2015/05/real-gif-posting-on-facebook/). *Wired*. wired.com. Retrieved 29 May 2015.

7. Perez, Sarah (29 May 2015). "Facebook Confirms It Will Officially Support GIFs" (https://techcrunch.com/2015/05/29/facebook-confirms-it-will-officially-support-gifs). techcrunch.com. Retrieved 29 May 2015.
8. "Introducing GIF Stickers" (https://instagram-press.com/blog/2018/01/23/introducing-gif-stickers/). *Instagram*. 23 January 2018. Retrieved 19 September 2019.
9. "Oxford Dictionaries USA Word of the Year 2012" (http://blog.oxforddictionaries.com/press-releases/us-word-of-the-year-2012/). *OxfordWords blog*. Oxford American Dictionaries. 13 November 2012. Retrieved 1 May 2013.
10. Flood, Alison (27 April 2013). "*Gif* is America's word of the year? Now that's what I call an omnishambles" (https://www.theguardian.com/books/booksblog/2012/nov/14/gif-america-word-year-omnishambles). Books blog. *The Guardian*. London. Retrieved 1 May 2013.
11. Olsen, Steve. "The GIF Pronunciation Page" (http://www.olsenhome.com/gif/). Retrieved 6 March 2009.
12. "Gif's inventor says ignore dictionaries and say 'Jif' " (https://www.bbc.co.uk/news/technology-22620473). *BBC News*. 22 May 2013. Retrieved 22 May 2013.
13. "Stack Overflow Developer Survey 2017" (https://insights.stackoverflow.com/survey/2017#work-_-how-do-you-pronounce-gif). 2017. Retrieved 19 August 2018.
14. "How do you pronounce "GIF"?" (https://www.economist.com/blogs/graphicdetail/2017/06/daily-chart-21). *The Economist*. Retrieved 4 January 2018.
15. "GIF" (http://dictionary.reference.com/browse/GIF). *The American Heritage Abbreviations Dictionary, Third Edition*. Houghton Mifflin Company. 2005. Retrieved 15 April 2007.
16. "GIF" (http://dictionary.cambridge.org/us/dictionary/business-english/gif?q=gif). *The Cambridge Dictionary of American English*. Cambridge University Press. Retrieved 19 February 2014.
17. "Gif - Definition from the Merriam-Webster Dictionary" (http://www.merriam-webster.com/dictionary/gif). *Merriam-Webster Dictionary*. Merriam-Webster, Incorporated. Retrieved 6 June 2013.
18. "GIF" (http://www.oxforddictionaries.com/definition/english/GIF). *Oxford Dictionaries Online*. Oxford University Press. Retrieved 7 October 2014.
19. *The New Oxford American Dictionary* (2nd ed.). Oxford University Press. 2005. p. 711.
20. *The New Oxford American Dictionary* (3rd ed.). 2012. (part of the Macintosh built-in dictionaries).
21. O'Leary, Amy (21 May 2013). "An Honor for the Creator of the GIF" (http://bits.blogs.nytimes.com/2013/05/21/an-honor-for-the-creator-of-the-gif/?smid=tw-nytimes). *The New York Times*. Retrieved 22 May 2013.
22. Rothberg, Daniel (4 December 2013). " 'Jeopardy' wades into 'GIF' pronunciation battle" (http://www.latimes.com/nation/shareitnow/la-sh-alex-trebek-gif-pronunciation-jeopardy-20131204,0,1162165.story#axzz2mYD3tX3M). *Los Angeles Times*. Retrieved 4 December 2013.
23. O'Leary, Amy (23 May 2013). "Battle Over 'GIF' Pronunciation Erupts" (http://bits.blogs.nytimes.com/2013/05/23/battle-over-gif-pronunciation-erupts/?_r=0). *The New York Times*.
24. Valinsky, Jordan (February 25, 2020). "Jif settles the great debate with a GIF peanut butter jar" (https://www.cnn.com/2020/02/25/business/jif-gif-peanut-butter-trnd/index.html). *CNN*. Retrieved February 25, 2020.
25. Marur, D.R.; Bhaskar, V. (March 2012). "Comparison of platform independent electronic document distribution techniques". *Devices, Circuits and Systems (ICDCS)*. International Conference on Devices, Circuits and Systems (ICDCS) (http://www.ieee.org/conferences_events/conferences/conferencedetails/index.htm?Conf_ID=19586). Karunya University; Coimbatore, India: IEEE. pp. 297–301. doi:10.1109/ICDCSyst.2012.6188724 (https://doi.org/10.1109%2FICDCSyst.2012.6188724). ISBN 9781457715457.
26. S. Chin; D. Iverson; O. Campesato; P. Trani (2011). *Pro Android Flash* (http://yuliana.lecturer.pens.ac.id/Android/Buku/Pro-Android-Flash.pdf) (PDF). New York: Apress. p. 350. ISBN 9781430232315. Retrieved 11 March 2015.

27. Andreas Kleinert (2007). "GIF 24 Bit (truecolor) extensions" (https://web.archive.org/web/20120316215949/http://uk.aminet.net/docs/misc/GIF24.readme). Archived from the original (http://uk.aminet.net/docs/misc/GIF24.readme) on 16 March 2012. Retrieved 23 March 2012.
28. Philip Howard. "True-Color GIF Example" (https://web.archive.org/web/20150222123613/http://phil.ipal.org/tc.html). Archived from the original (http://phil.ipal.org/tc.html) on 22 February 2015. Retrieved 23 March 2012.
29. "Nullsleep - Jeremiah Johnson - Animated GIF Minimum Frame Delay Browser Compatibility Study" (https://nullsleep.tumblr.com/post/16524517190/animated-gif-minimum-frame-delay-browser-compatibility). Retrieved 26 May 2015.
30. "They're different! How to match the animation rate of gif files accross [*sic*] browsers" (https://web.archive.org/web/20170201034945/http://blog.fenrir-inc.com/us/2012/02/theyre-different-how-to-match-the-animation-rate-of-gif-files-accross-browsers.html). *Developer's Blog*. 14 February 2012. Archived from the original (http://blog.fenrir-inc.com/us/2012/02/theyre-different-how-to-match-the-animation-rate-of-gif-files-accross-browsers.html) on 1 February 2017. Retrieved 15 June 2017.
31. Royal Frazier. "All About GIF89a" (https://web.archive.org/web/19990418091037/http://www6.uniovi.es/gifanim/gifabout.htm). Archived from the original (http://www6.uniovi.es/gifanim/gifabout.htm) on 18 April 1999. Retrieved 7 January 2013.
32. Scott Walter (1996). *Web Scripting Secret Weapons*. Que Publishing. ISBN 0-7897-0947-3.
33. Law, Eric (7 June 2010). "Trivia: Animated GIF Timing" (https://blogs.msdn.microsoft.com/ieinternals/2010/06/07/trivia-animated-gif-timing/). *MSDN*. Microsoft. Retrieved 30 November 2018.
34. "XMP Specification Part 3: Storage in Files" (https://wwwimages2.adobe.com/content/dam/acom/en/devnet/xmp/pdfs/XMP%20SDK%20Release%20cc-2016-08/XMPSpecificationPart3.pdf) (PDF). Adobe. 2016. pp. 11–12. Retrieved 16 August 2018.
35. Greg Roelofs. "History of the Portable Network Graphics (PNG) Format" (http://www.libpng.org/pub/png/pnghist.html). Retrieved 23 March 2012.
36. Stuart Caie. "Sad day... GIF patent dead at 20" (http://www.kyzer.me.uk/essays/giflzw/). Retrieved 23 March 2012.
37. U.S. Patent 4,558,302 (https://www.google.com/patents/US4558302)
38. "The GIF Controversy: A Software Developer's Perspective" (https://mike.pub/19950127-gif-lzw). Retrieved 26 May 2015.
39. "Unisys Clarifies Policy Regarding Patent Use in On-Line Service Offerings" (https://web.archive.org/web/20070207110047/http://lpf.ai.mit.edu/Patents/Gif/unisys.html). Archived from the original (https://www.thefreelibrary.com/UNISYS+CLARIFIES+POLICY+REGARDING+PATENT+USE+IN+ON-LINE+SERVICE+...-a016000999) on 7 February 2007. – archived by League for Programming Freedom
40. "Libungif" (https://directory.fsf.org/wiki/Libungif). Retrieved 26 May 2015.
41. Cargill, Tom (1 October 2001). "Replacing a Dictionary with a Square Root" (http://www.drdobbs.com/architecture-and-design/replacing-a-dictionary-with-a-square-roo/184404817). *Dr. Dobb's Journal*. Retrieved 20 January 2017.
42. "LZW Software and Patent Information" (https://web.archive.org/web/20090608194513/http://www.unisys.com/about__unisys/lzw/lzw__license__english.htm). Archived from the original (http://www.unisys.com/about__unisys/lzw/lzw__license__english.htm) on 8 June 2009. Retrieved 31 January 2007. – clarification of 2 September 1999
43. Unisys Not Suing (most) Webmasters for Using GIFs (https://slashdot.org/story/99/08/31/0143246/unisys-not-suing-most-webmasters-for-using-gifs) – Slashdot investigation into the controversy
44. "Burn All GIFs Day" (https://web.archive.org/web/19991013083423/http://burnallgifs.org/). Archived from the original (http://burnallgifs.org/) on 13 October 1999.
45. Burn All GIFs (http://burnallgifs.org/archives/) – A project of the League for Programming Freedom (latest version)

46. "License Information on GIF and Other LZW-based Technologies" (https://web.archive.org/web/20090602212118/http://www.unisys.com/about__unisys/lzw). Archived from the original (http://www.unisys.com/about__unisys/lzw) on 2 June 2009. Retrieved 26 April 2005.
47. "Why There Are No GIF Files on GNU Web Pages" (https://www.gnu.org/philosophy/gif.html). Free Software Foundation. Retrieved 19 May 2012.
48. "PNG versus GIF Compression" (http://www.websiteoptimization.com/speed/tweak/png/). Retrieved 8 June 2009.
49. "AlphaImageLoader Filter" (http://msdn.microsoft.com/en-us/library/ms532969(VS.85).aspx). Microsoft. Retrieved 26 May 2015.
50. "What's New in Internet Explorer 7" (http://msdn.microsoft.com/en-us/library/ms649487%28VS.85%29.aspx). *MSDN*. Retrieved 6 March 2009.
51. "PNG Image File Format" (http://www.scantips.com/basics9p.html). Retrieved 8 June 2009.
52. "Can I use... Support tables for HTML5, CSS3, etc" (https://caniuse.com/#feat=apng). *caniuse.com*.
53. "VOTE FAILED: APNG 20070405a" (http://sourceforge.net/mailarchive/message.php?msg_id=131482). SourceForge mailing list. 20 April 2007.
54. "Discussion for a simple "animated" PNG format" (https://web.archive.org/web/20090226103407/http://gjuyn.xs4all.nl/pnganim.html). Archived from the original (http://gjuyn.xs4all.nl/pnganim.html) on 26 February 2009. Retrieved 12 July 2011.
55. "APNG Specification" (https://wiki.mozilla.org/APNG_Specification). Retrieved 26 May 2015.
56. Mozilla Labs » Blog Archive » Better animations in Firefox 3 (https://blog.mozilla.org/labs/2007/08/better-animations-in-firefox-3/)
57. "195280 – Removal of MNG/JNG support" (https://bugzilla.mozilla.org/show_bug.cgi?id=195280). Retrieved 26 May 2015.
58. "18574 – (mng) restore support for MNG animation format and JNG image format" (https://bugzilla.mozilla.org/show_bug.cgi?id=18574). Retrieved 26 May 2015.
59. "Chromium Blog: Chrome 32 Beta: Animated WebP images and faster Chrome for Android touch input" (https://blog.chromium.org/2013/11/chrome-32-beta-animated-webp-images-and.html). Blog.chromium.org. 21 November 2013. Retrieved 1 February 2014.
60. "Introducing GIFV - Imgur Blog" (http://imgur.com/blog/2014/10/09/introducing-gifv/). imgur.com. 9 October 2014. Retrieved 14 December 2014.
61. Thomson, Gavin; Shah, Athar (2017). "Introducing HEIF and HEVC" (https://devstreaming-cdn.apple.com/videos/wwdc/2017/503i6plfvfi7o3222/503/503_introducing_heif_and_hevc.pdf) (PDF). Apple Inc. Retrieved 5 August 2019.
62. "HEIF Comparison - High Efficiency Image File Format" (https://nokiatech.github.io/heif/comparison.html). Nokia Technologies. Retrieved 5 August 2019.
63. "#3271 (Allow using additional pixel formats with libvpx-vp9) – FFmpeg" (https://trac.ffmpeg.org/ticket/3271?cversion=0&cnum_hist=3). *trac.ffmpeg.org*.
64. Dewey, Caitlin. "Meet the technology that could make GIFs obsolete" (https://www.washingtonpost.com/blogs/style-blog/wp/2014/04/08/meet-the-technology-that-could-make-gifs-obsolete/). *The Washington Post*. Retrieved 4 February 2015.
65. "WebM support on 4chan" (http://blog.4chan.org/post/81896300203/webm-support-on-4chan). 4chan Blog. Retrieved 4 February 2015.
66. "Introducing GIFV" (http://blog.imgur.com/2014/10/09/introducing-gifv/). Imgur. 9 August 2014. Retrieved 21 July 2016.
67. Allan, Patrick. "Imgur Revamps GIFs for Faster Speeds and Higher Quality with GIFV" (https://lifehacker.com/imgur-revamps-gifs-for-faster-speeds-and-higher-quality-1644494212). Lifehacker. Retrieved 4 February 2015.
68. "GIF Revolution" (https://telegram.org/blog/gif-revolution). *Official Telegram Blog*. Retrieved 4 January 2016.

External links

- The GIFLIB project (http://giflib.sourceforge.net/)
- spec-gif89a.txt (https://www.w3.org/Graphics/GIF/spec-gif89a.txt) GIF 89a specification on w3.org
- GIF 89a specification reformatted into HTML (https://web.archive.org/web/20160304075538/http://qalle.net/gif89a.php)
- LZW and GIF explained (https://www.eecis.udel.edu/~amer/CISC651/lzw.and.gif.explained.html)
- Animated GIFs (https://www.pbs.org/video/2207348428/): a six-minute documentary produced by Off Book (web series)
- GifCities (https://gifcities.org) (The GeoCities Animated GIF Search Engine)