

The `animategif` package

Animated GIFs in PDF documents, straight from the `.gif` file

Ali Ramlaoui

<https://github.com/Ramlaoui/animategif>

Version 1.0.0, 2026-10-06

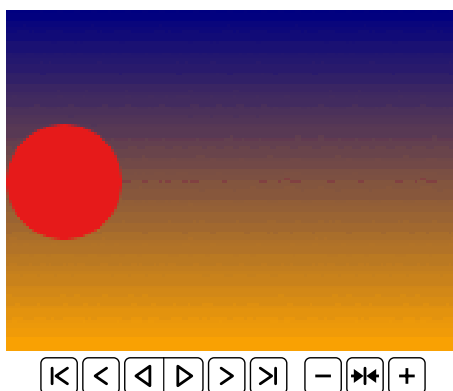
Abstract

`animategif` lets you write `\animategif{movie.gif}` and get the animation in your PDF. It decodes the GIF itself, in pure Lua, and hands the frames to the `animate` package. No ImageMagick or other external programs, and no frame extraction by hand. It honours frame delays, disposal methods, transparency, interlacing and loop counts, encodes frames as compact PNGs, stores only the pixels that change between frames, and caches the result.

1 Quick start

```
\documentclass{article}
\usepackage{animategif}
\begin{document}
\animategif[width=6cm]{animategif-demo}    % .gif is optional
\end{document}
```

Compile with `lualatex`, or with `pdflatex -shell-escape` or `xelatex -shell-escape`. Here is the result. In a viewer that plays animate animations (see §2) the circle moves. Elsewhere you see the first frame.



2 What it does, and what it does not

PDF has no animated image type, so every animated PDF relies on the viewer's JavaScript. `animate` builds the animation from a sequence of images that it switches with JavaScript, and `animategif` supplies those images from a GIF. The animations therefore play in the same viewers

as any `animate` animation: Adobe Acrobat/Reader, KDE Okular, PDF-XChange and Foxit Reader. Other viewers (Apple Preview, Skim, and the built-in viewers of web browsers) show a still poster frame, by default the first one; choose another with `animate`'s `poster` option, e.g. `poster=last`.

Without `animategif`, the usual route is to extract the frames with an external program (`magick in.gif -coalesce frame-%d.png`), work out the frame rate, and call `\animategraphics`. `animategif` does all of this inside the TeX run. It also respects per-frame delays, which a single frame rate cannot, and stores much smaller frames.

3 Engines and the cache

The decoder is the Lua module `animategif.lua`.

- **LuaLaTeX** runs it in-process. Nothing else is needed.
- **pdfLaTeX** and **XeLaTeX** run it as `texlua animategif.lua ...` through shell escape, so they need `-shell-escape`. Restricted shell escape (the TeX Live default) is not enough.

Decoded frames are written to `animategif-cache/<name>-<hash>/<settings>/` where `<hash>` identifies the GIF's contents. Later runs reuse the cache, and changing the GIF or the frame options (`frames`, `step`, `optimize`, `keyframe`, `downsample`) decodes it again. Options such as `speed`, `fps`, `plays` or any `animate` option take effect without decoding again.

Because the cache is plain files, a document whose cache exists compiles with any engine and *without* shell escape. This is useful for journals, arXiv or co-authors: compile once with LuaLaTeX and ship the `animategif-cache` directory with the sources. If a GIF is not in the cache and cannot be decoded, the error message gives the command that fills the cache, which you can also run yourself:

```
texlua $(kpsewhich animategif.lua) frames movie.gif <dir> [key=value ...]
```

Delete `animategif-cache` at any time to reclaim space, and add it to `.gitignore` unless you want to ship it.

4 Options

Options are given to `\animategif`, to the package, or to `\animategifsetup{<options>}`, which sets them for the rest of the current group. Any option that `animategif` does not know is passed to `animate`. That includes `width`, `height`, `scale`, `controls`, `poster`, `palindrome`, `autopause`, `autoresume`, `method` and `alttext`; see the `animate` manual. Frames are numbered from 0, as in `animate`.

Option	Default	Meaning
<code>frames=<a>-</code>	0-	Only use frames <i><a></i> to <i></i> . <i><a></i> - runs to the end, <i>-</i> starts at the beginning, and <i><n></i> is a single frame.
<code>step=<n></code>	1	Keep every <i><n></i> th frame of the range. A dropped frame's delay goes to the kept frame before it, so the timing is unchanged.
<code>speed=<factor></code> <code>fps=<rate></code>	1	Play faster (2) or slower (0.5). Ignore the GIF's delays and play at a constant frame rate (times speed).
<code>plays=<n></code>	auto	How many times to play: auto follows the GIF's loop count (as browsers do), forever loops, and a number plays that many times and stops on the last frame.
<code>still[=<frame>]</code>	false	Include one frame as an ordinary image instead of an animation: first , last or a frame number. Of the options, only width , height , scale , angle , trim , clip and the like are used.
<code>handout=still animate</code>	still	In beamer's handout mode, GIFs are included as still images unless handout=animate .
<code>label=<name></code>		Name for animate 's JavaScript interface (<code>anim['<name>']</code>), e.g. to control playback with buttons.
<code>downsample=<n></code>	1	Reduce the resolution by an integer factor, averaging $n \times n$ blocks. This shrinks big GIFs a lot.
<code>optimize=<bool></code>	true	Store only the pixels that change between frames, stacked on the previous ones (see §7). false stores every frame in full.
<code>keyframe=<n></code>	30	With optimize , store a full frame at least every <i><n></i> frames.
<code>cachedir=<dir></code>	animategif-cache	Where decoded frames go.

5 Examples

```
% half speed, with a control bar, last frame as the poster for other viewers
\animategif[speed=0.5, controls, poster=last]{demo}
```

```
% frames 10 to 40, every other one, played three times
\animategif[frames=10-40, step=2, plays=3, width=\linewidth]{demo}
```

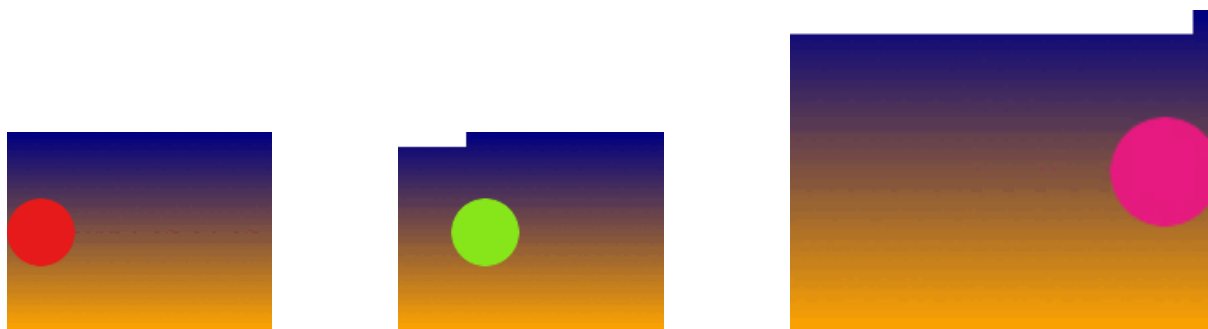
```
% a big screen recording at half resolution
\animategif[downsample=2, width=\textwidth]{recording}
```

```
% the final state, as a plain image (e.g. for print)
\animategif[still=last, width=4cm]{demo}
```

```
% all GIFs as still images, e.g. for a printed version
\animategifsetup{still}
```

Three of these, each with `width=3.5cm`: `speed=0.5`, `frames=4-11` with `plays=2`, and

still=last:



In beamer, `\animategif` works like any `animate` animation. The `animate` manual explains how to combine animations with overlays.

6 The command-line tool

`animategif.lua` also works on its own with `texlua`:

```
texlua animategif.lua info movie.gif
texlua animategif.lua frames movie.gif <dir> [key=value ...]
texlua animategif.lua still movie.gif <dir> [frame=<n>] [downsample=<n>]
```

`info` prints the size, the number of frames, the duration and the loop count. `frames` writes what `\animategif` uses: the PNG images and `info.tex`. It takes `optimize`, `keyframe`, `first`, `last`, `step` and `downsample`. `still` writes one composited frame; a negative `frame` counts from the end. Set the environment variable `ANIMATEGIF_PROFILE=1` to see where the time goes.

7 How it works

The decoder reads the GIF stream and decompresses the LZW image data. It composites each frame onto a canvas the size of the logical screen, following the frame's disposal method (keep, restore to background, restore to previous), its transparent colour, interlacing, and local or global palettes. Frames that extend past the logical screen are clipped. Delays of 0 or 1 hundredths of a second are played at 10, as browsers do. A GIF without a loop extension plays once. A loop count of $\langle n \rangle$ plays $n + 1$ times, and 0 loops forever.

Each frame is written as a PNG. The PNG is indexed (one byte per pixel) when the frame has at most 256 colours, as GIF frames do, and RGB(A) otherwise.

With `optimize`, a frame stores only the pixels that differ from the previous frame. The others are transparent, and `animate`'s timeline stacks these delta images on the last full frame. Full keyframes are written for the first frame, after `keyframe` frames, when pixels become transparent (a delta can only paint), and whenever a delta would not be clearly smaller. Identical consecutive frames cost nothing. Around the changed pixels, the transparent pixels carry the colour that shows through them, so that viewers which smooth downscaled images do not draw seams at the edges of the changes.

`animategif` then writes an `animate` timeline that gives each frame its own duration and stacks the delta images, and calls `\animategraphics` with it. A finite number of plays is written into the

timeline as repeats of the frame sequence. The images are shared, so this costs almost nothing, and playback stops on the last frame without JavaScript tricks.

8 Limitations

- Animations need a JavaScript-capable viewer (§2).
- Every distinct frame is an image in the PDF. Long or large GIFs make big PDFs. `frames`, `step` and `downsample` help.
- Decoding is pure Lua: roughly a second per 5 million pixels of animation. It happens once per GIF and settings, thanks to the cache.
- pdfLaTeX and XeLaTeX need unrestricted shell escape, or a filled cache.

9 License

Copyright © 2026 Ali Ramlaoui. This work may be distributed and/or modified under the conditions of the L^AT_EX Project Public License, version 1.3c or later. It has the LPPL maintenance status “maintained”. Bug reports and contributions are welcome at <https://github.com/Ramlaoui/animategif>.

10 Change history

1.0.0 (2026-10-06) First public release.