The animate Package

Alexander Grahn *
a.grahn@web.de

26th January 2009

Abstract

A LaTeX package for creating portable, JavaScript driven PDF animations from sets of vector graphics or rasterized image files or from inline graphics.

Keywords: include portable PDF animation animated PDF animating embed animated graphics LaTeX pdfLaTeX PSTricks pgf TikZ LaTeX-picture MetaPost inline graphics vector graphics animated GIF LaTeX dvips ps2pdf dvipdfmx XeLaTeX JavaScript Adobe Reader

Contents

*Animated GIF taken from phpBB forum software and burst into a set of EPS files using ImageMagick before embedding.

1 Introduction

This package provides an interface to create PDFs with animated content from sets of graphics or image files, from inline graphics, such as LaTeX-picture, PSTricks or pgf/TikZ generated pictures, or just from typeset text. Unlike standard movie/video formats, which can be embedded, for example, using the LaTeX package 'movie15' [4], package 'animate' allows for animating vector graphics. The result is roughly similar to the SWF (Flash) format, although not as space-efficient.

Package 'animate' supports the usual PDF making workflows, i. e. pdfLaTeX, LaTeX → `dvips` → `ps2pdf`/Distiller and (XeH)LaTeX → `(x)dvipdfmx`.

The final PDF can be viewed in current Adobe Reader®s on all supported platforms.

2 Requirements

ε-TeX

pdfTeX, version ≥ 1.20 for direct PDF output

Ghostscript, version ≥ 8.31 or Adobe Distiller for PS to PDF conversion

dvipdfmx, version ≥ 20080607 for DVI to PDF conversion

Adobe Reader, version ≥ 6

3 Installation

Unzip the file 'animate.tds.zip' into the local TDS root directory which can be found by running '`kpsewhich -var-value TEXMFLOCAL`' on the command line.

After installation, update the filename database by running '`texhash`' on the command line.

MiKTeX users should run the package manager for installation.

4 Using the package

First of all, read Section 8 on problems related to this package. Then, invoke the package by putting the line

```
\usepackage[<package options>]{animate}
```

to the preamble of your document, i. e. somewhere between `\documentclass` and `\begin{document}`.

'animate' honours the package options:

```
dvipdfmx
xetex
autoplay
autopause
autoresume
loop
palindrome
draft
final
controls
buttonsize=<size>
buttonbg=<colour>
buttonfg=<colour>
step
useocg
poster[=first | none | last]
```

Except for '`dvipdfmx`' and '`xetex`', the options above are also available (among others) as command options and will be explained shortly. However, if used as package options they have global scope, taking effect on all animations in the document. In turn, command options locally override global settings. Options without an argument are boolean options and can be negated, with the exception of package-only options '`dvipdfmx`' and '`xetex`', by appending '`=false`'.

If PDF is generated via DVI and Postscript by the command sequence `latex` → `dvips` → `ps2pdf`, the 'graphicx' package is required. *Important:* The `dvips` option '`-Ppdf`' should *not* be set when converting the intermediate DVI into Postscript. If you cannot do without, put '`-X 2400 -Y 2400`' *after* '`-Ppdf`' on the command line. Sometimes, if the same animation sequence made from graphics files is to be embedded multiple times into the document, more compact PDF output may be obtained by passing option '`-dMaxInlineImageSize=0`' (Unix) or '`-dMaxInlineImageSize#0`' (Win/DOS) to `ps2pdf`.

For '`dvipdfmx`' and XeLaTeX, the 'graphicx' package is required and corresponding package options '`dvipdfmx`' or '`xetex`' must be set for 'animate' and 'graphicx'.

Occasionally, a second LaTeX run may be necessary to resolve internally created object references. Appropriate warnings will be issued in such cases.

5 The user interface

Package 'animate' provides the command

```
\animategraphics[<options>]{<frame rate>}{<file basename>}{<first>}{<last>}
```

and the environment

```
\begin{animateinline}[<options>]{<frame rate>}
    ... typeset material ...
\newframe[<frame rate>]
```

```
    ... typeset material ...
\newframe*[<frame rate>]
    ... typeset material ...
\newframe
\multiframe{<number of frames>}{[<variables>]}{
  ... repeated (parameterized) material ...
}
\end{animateinline}
```

While `\animategraphics` can be used to assemble animations from sets of existing graphics files or from multipage PDF, the environment '`animateinline`' is meant to create the animation from the typeset material it encloses. This material can be pictures drawn within the LaTeX '`picture`' environment or using the advanced capabilities of PSTricks or pgf/TikZ. Even ordinary textual material may be animated in this way. The parameter `<frame rate>` specifies the number of frames per second of the animation.

The command `\newframe` terminates a frame and starts the next one. It can be used inside the '`animateinline`' environment only. There is a starred variant, `\newframe*`. If placed after a particular frame it causes the animation to pause at that frame. The animation continues normally after clicking it again. Both `\newframe` variants take an optional argument that allows to change the frame rate in the middle of an animation.

The command `\multiframe` allows to build loops around pictures. The first argument `<number of frames>` does what one would expect it to do, the second argument `<variables>` is a comma-separated list of variable declarations. The list may be of arbitrary, even zero, length. Variables may be used to parameterize pictures which are defined in the loop body (third argument of `\multiframe`). A single variable declaration has the form

```
<variable name>=<initial value>+<increment>
```

`<variable name>` is a sequence of one or more letters *without* a leading backslash[1]. The first (and possibly only) letter of the variable name determines the type of the variable. There are three different types: integers ('`i`', '`I`'), reals ('`n`', '`N`', '`r`', '`R`') and dimensions or LaTeX lengths ('`d`', '`D`'). Upon first execution of the loop body, the variable takes the value `<initial value>`. Each further iteration increments the variable by `<increment>`. Negative increments must be preceded by '`-`'. Here are some examples: '`i=1+2`', '`Rx=10.0+-2.25`', '`dim=20pt+1ex`'. Within the loop body, variables are expanded to their current value by prepending a backslash to the variable name, that is `\i`, `\Rx` and `\dim` according to the previous examples. `\multiframe` must be surrounded by `\begin{animateinline}` and `\end{animateinline}` or by any of the `\newframe` variants. Two consecutive `\multiframe` commands must be separated by one of the `\newframe` variants.

By default, the animation is built frame by frame in the order of inclusion of the embedded material. However, extended control of the order of appearance,

[1] This is different from `\multido` (package 'multido') where variable names have a leading '\' in the declaration.

superposition and repetition of the material is available through the '`timeline`' option (see Section 6.2).

Sets of graphics files

All files of the sequence should exist and be consecutively numbered. (Exception to this rule is allowed in connection with the '`every`' option, see below.) `<file basename>` is the leftmost part of the file name that is common to all members of the sequence. `<first>` is the number of the first and `<last>` the number of the last file in the set. File names may be simply numbered, such as 0 . . . 99. If there are leading zeros, make sure that all file numbers have the same number of digits, such as 0000 . . . 0099.

There is no file name extension to be specified as a parameter. The possible file formats depend on the output driver being used. In the case of LaTeX+`dvips`, files with the extension 'eps' are searched for at first, followed by 'mps' (MetaPost-generated Postscript) and 'ps'. With pdfLaTeX the searching order is: (1) 'pdf', (2) 'mps', (3) 'png', (4) 'jpg', (5) 'jpeg', (6) 'jbig2', (7) 'jb2', (8) 'jp2'[1], (9) 'j2k'[1], (10) 'jpx'[1] and with XeLaTeX or LaTeX+`dvipdfmx`: (1) 'pdf', (2) 'mps', (3) 'eps', (4) 'ps', (5) 'png', (6) 'jpg', (7) 'jpeg', (8) 'bmp'. That is, files capable to store vector graphics are found first. Make sure that all file names have *lower case* extensions.

For example, given the sequence 'frame_5.png' through 'frame_50.png' from a possibly larger set that shall be used to build an animation running at 12 frames per second. Then, the correct inclusion command reads

```
\animategraphics{12}{frame_}{5}{50}
```

Multipage PDF (pdfLaTeX, XeLaTeX) *and JBIG2* (pdfLaTeX) *inclusion*

If the file '`<file basename>`.(pdf|jbig2|jb2)' exists (again, there is no file name extension to be specified), it is taken as a multipage document where each page represents one frame of the animation. In this case, the last two arguments, `<first>` & `<last>`, are interpreted differently from above; they specify a zero-based range of pages to be included in the animation. Either or both of them may be left empty, '`{}`', in which case they default to 0 and $n-1$, where n is the total number of available pages. Arguments that fall outside this range are automatically corrected to the maximum allowed number, i. e. $n-1$.

For example, the line

```
\animategraphics{12}{frames}{}{}
```

would create an animation from all pages of the file 'frames.pdf', running at 12 fps.

6 Command options

The following options to `\animategraphics` and '`animateinline`' have been provided:

[1] JPEG2000 is not yet supported by pdfTeX.

6.1 Basic options

`poster[=first | none | last]`

Specifies which frame (first, last or none) to display and print if the animation is not activated. The first frame is shown by default. Thus '`poster`' or '`poster=first`' need not be explicitly set.

`every=<num>`

Build animation from every `<num>`*th* frame only. Skipped frames are discarded and not embedded into the document. In the case of `\animategraphics`, skipped input files may be missing.

`autopause`

Pause animation when the page is closed, instead of stopping and rewinding it to the default frame.

`autoplay`

Start animation after the page has opened. Also resumes playback of a previously paused animation.

`autoresume`

Resume previously paused animation when the page is opened again.

`loop`

The animation restarts immediately after reaching the end.

`palindrome`

The animation continuously plays forward and backward.

`step`

Step through the animation by one frame per mouse-click.

`width=<width>`
`height=<height>`
`depth=<depth>`

Resize the animation widget. Option '`depth`' specifies how far the animation widget should extend below the bottom line of the running text. If only one or two of these options are given, the remaining, unspecified dimensions are scaled to keep the aspect ratio. Any valid TEX dimension is accepted as a parameter. Moreover, the length commands `\width`, `\height`, `\depth` and `\totalheight` may be used to refer to the original dimensions of the animation widget which correspond to the size of the first frame of the animated sequence.

`scale=<factor>`

Scales the animation widget by `<factor>`.

`bb=<llx> <lly> <urx> <ury>`

(`\animategraphics` only, requires package 'graphicx'.) The four, space separated arguments set the bounding box of the graphics files. Units can be omitted, in which case 'bp' (Postscript points) is assumed.

`viewport=<llx> <lly> <urx> <ury>`

(`\animategraphics` only, requires package 'graphicx'.) This option takes four arguments, just like '`bb`'. However, in this case the values are taken relative to the origin specified by the bounding box in the graphics files.

`trim=<left> <bottom> <right> <top>`

(`\animategraphics` only, requires package 'graphicx'.) Crops graphics at the edges. The four lengths specify the amount to be removed from or, if negative values have been provided, to be added to each side of the graphics.

`controls`

Inserts control buttons below the animation widget. The meaning of the buttons is as follows, from left to right: stop & first frame, step backward, play backward, play forward, step forward, stop & last frame, decrease speed, default speed, increase speed. Both 'play' buttons are replaced by a large 'pause' button while the animation is playing.

`buttonsize=<size>`

Changes the control button height to `<size>`, which must be a valid TEX dimension. The default button height is `1.44em` and thus scales with the current font size.

`buttonbg=<colour>`
`buttonfg=<colour>`

By default, control button widgets are drawn with black strokes on transparent background. The background can be turned into a solid colour by the first option, while the second option specifies the stroke colour. The parameter `<colour>` is an array of colon-(`:`)-separated numbers in the range from 0.0 to 1.0. The number of array elements determines the colour model in which the colour is defined: (1) gray value, (3) RGB, (4) CMYK. For example, '`1`', '`1:0.5:0.2`' and '`0.5:0.3:0.7:0.1`' are valid colour specifications.

`draft`
`final`

With '`draft`' the animation is not embedded. Instead, a box with the exact dimensions of the animation is inserted. Option '`final`' does the opposite as it forces the animation to be built and embedded. Both options can be used to reduce compilation time during authoring of a document. To get the most out of them it is recommended to set '`draft`' globally as a package or class option and to set '`final`' locally as a command option of the animation that is currently worked on. After the document has been finished, the global '`draft`' option can be removed to embed all animations.

`useocg`

Use an alternative animation method based on Optional Content Groups (OCGs, also known as PDF Layers). May result in less performing animations.

`measure`

Measures the frame rate during one cycle of the animation. (For testing purposes.)

`begin={<begin text>}`
`end={<end text>}`

('`animateinline`' only.) `<begin text>` and `<end text>` are inserted into the code at start and end of each frame. Mainly used for setting up some drawing environment, such as

```
begin={\begin{pspicture}(...)(...)}, end={\end{pspicture}}
```

6.2 The '`timeline`' option

`timeline=<timeline file>`

`<timeline file>` is a plain text file whose contents determines the order of appearance of the embedded material during the animation. It allows to freely rearrange, repeat and overlay the material at any point of the animation. This may greatly reduce the file size of the resulting PDF, as objects that do not change between several or all frames, such as coordinate axes or labels, can be embedded once and re-used in other frames of the animation.

If a timeline is associated with the animation, the graphics files or inline graphics embedded by `\animategraphics` and '`animateinline`' no longer represent the actual frames of the animation. Rather, they are a collection of *transparencies* that can be played with at will. However, it is now up to the author's responsability to construct a timeline that makes use of *each* of those transparencies and to put them into a sensible order. In order to identify the transparencies within the timeline file, they are numbered in the order of their inclusion, starting at zero.

Each line of the timeline file that is not blank and which does not begin with a comment ('`%`') specifies *one* frame of the animation. There may be more transparencies than animation frames and vice-versa. A frame specification consists of three, colon-(`:`)-separated fields:

```
[*]:[<frame rate>]:[<frame content>]
```

While any field may be left blank, the colons are mandatory.

An asterisk ('`*`') in the leftmost field causes the animation to pause at that frame, very much as a `\newframe*` would do; a number in the second field changes the frame rate of the animation section that follows. (In connection with the '`timeline`' option the asterisk extension and the optional `<frame rate>` argument of `\newframe` cease to make sense and will be tacitly ignored, if present.) The third field `<frame content>` is a comma-separated *list* of *transparency specifications* that determines the content of the frame. A *single* transparency specification obeys the syntax

```
<transparency ID>[x<number of frames>]
```

where `<transparency ID>` is an integer number that identifies the transparency to be drawn into the current animation frame. As pointed out above, the transparencies are consecutively numbered in the order of their inclusion, starting at zero. The optional postfix '`x<number of frames>`' specifies the number of consecutive frames the transparency is to appear within. If omitted, a postfix of '`x1`' is assumed, which causes the transparency to be shown in the current frame only. Obviously, `<number of frames>` must be a non-negative integer number. The meaning of postfix '`x0`' is special; it causes the transparency to be shown in all frames, starting with the current one, until the end of the animation.

Note that the order in which transparency specifications appear in the timeline file determines their *depth* level. If a frame is composed of more than one transparency, transparency specifications on the left of the input line are closer to the background and will be overprinted by those on the right which are closer to the foreground. That is, the depth *de*creases from left to right within `<frame content>`. Also, if there are transparency specifications which span several frames (using postfix '`x<number of frames>`'), they will be overprinted by transparency specifications that appear on subsequent lines in the timeline file. That is, the depth decreases in top-down direction within the timeline file.

Consider the two timelines

```
::1,0 % zeroth transparency always in the foreground
::2,0
::3,0
::4,0
etc...
```

and

```
::0x0,1 % zeroth transparency put to the background
::2
::3
::4
etc...
```

In the first example, transparency No. 0 appears in the foreground throughout the animation; it will never be obscured by other transparencies' content. In the second example it is put to the background, because it is the first transparency specification in the file, and stays there for the rest of the animation.

When designing the timeline, care should be taken not to include a transparency more than once into the *same* animation frame. Besides the useless redundancy, this may slow down the animation speed in the Reader, because the graphical objects of a multiply included transparency have to be rendered unnecessarily often at the same time. 'animate' is smart enough to detect multiple inclusion and issues a warning message along with the transparency ID and the frame number if it occurs. Here is an example of a poorly designed timeline:

```
::0
::1x0
::2
::3
::4,2
::5,1 % bad: transparency '1' included twice
::6
```

Also, 'animate' finds and lists transparencies that have never been used in an animation timeline. This may help to avoid dead code in the final PDF.

Grouping objects into layers using ';'

The stack-like concept of animations, where transparencies are always put on top such that they overprint the content of previously deposited transparencies, can be inconvenient in certain situations. For example, it might be desirable to allow for changing the background image in the middle of an animation without affecting objects that are located in the foreground. For this purpose, transparency specifications can be grouped into *layers* using the semicolon (`;`) as a separator instead of the comma. This is best illustrated by an example:

```
% <--layer 1--> <--layer 2-->
%
::    0x49     ;   2x0,3x0    % transparency '0' used as background
::             ;   4x0,5x0    % image during the first 49 frames
::             ;   6x0,7x0
etc...
::             ;  98x0,99x0
::    1x0      ; 100x0,101x0  % transparency '1' used as new background
::             ; 102x0,103x0  % image until end of animation
::             ; 104x0,105x0
etc...
```

In this timeline, the transparencies are grouped into two layers. One is reserved for the background images, transparencies No. 0 & 1, to be exchanged after 49 frames, and another one for the foreground objects that are successively added to the scene. As can be seen in this example, layers need not be explicitly populated; the leading semicolons just ensure the proper assignment of transparencies to animation layers. Note that *without* setting up two layers, that is, by replacing the semicolons with commas, the foreground objects (transparencies 2 through 99) which have been added during the first 49 frames would be overprinted by the new background image, transparency 1, from frame 50 onward.

See the second animation, Fig. 2, in Section 7.1 for a working example that makes use of the timeline and the layer concept.

7 Examples

7.1 Animations from sets of files, using `\animategraphics` command

Animations in this section are made from graphics files that were prepared with METAPOST. Run '`mpost --tex=latex`' on the files ending in '.mp' in the 'doc/files' directory to generate the graphics files. Both examples make use of the '`timeline`' option to reduce the resulting PDF file size.

The first example, Fig. 1, originally written by Jan Holeček [3], shows the exponential function $y = e^x$ and its approximation by Taylor polynomials of different degree.

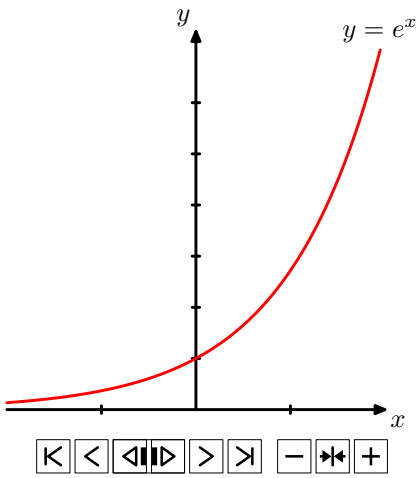

Figure 1

```
\documentclass{article}
\usepackage{animate}
\usepackage{graphics}

\begin{document}

\begin{center}
  \animategraphics[
    controls, loop,
    timeline=timeline.txt
  ]{4}{exp_}{0}{8}
\end{center}

\end{document}
```

Contents of file 'timeline.txt':

```
::0x0 % coordinate system & y=e^x, repeated until last frame
::1   % one blue curve per frame
::2
::3
::4
::5
::6
::7
::8
```

The second, somewhat more complex example, Fig. 2, animates the geometric construction of a scarabaeus. In addition to the use of a timeline, it introduces the layer concept. This example is adapted from Maxime Chupin's original METAPOST source file [1]. The present version separates stationary from moving parts of the drawing and saves them into different files. A total of 254 files, scarab_0.mps through scarab_253.mps, is written out by running '`mpost --tex=latex`' on the source file 'scarab.mp'. Files 0 through 100 contain the red line segments that make up the growing scarabaeus. Files 101 through

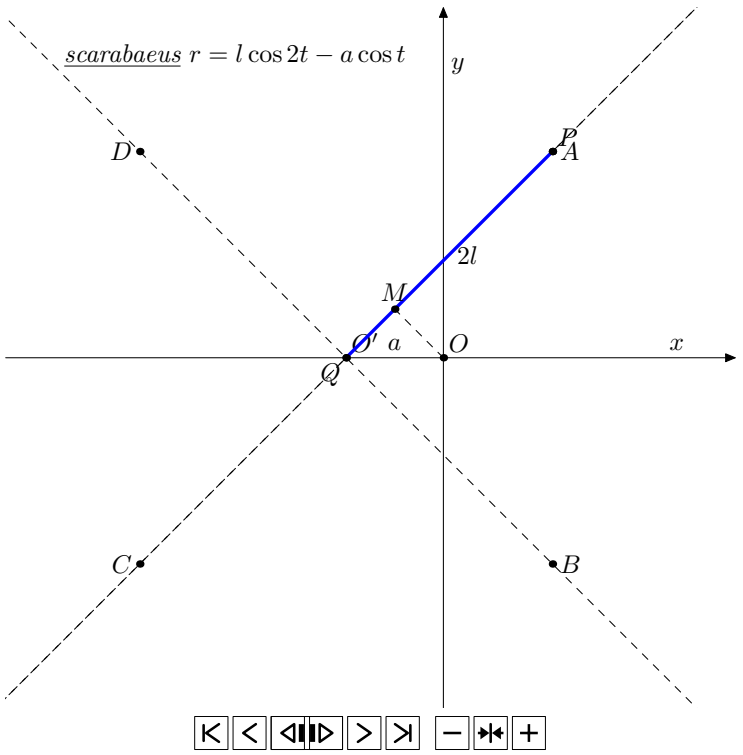

Figure 2

201 contain the moving construction lines and files 202 through 252 contain the gray lines which represent intermediate stages of the construction. The last file, No. 253, contains the coordinate axes, two stationary construction lines and the labels which do not move. A timeline file 'scarab.tln' is written out on-the-fly during the LaTeX run. It arranges the animation into three layers, forcing the gray lines into the background, the coordinate axes into the intermediate layer and the scarabaeus along with the moving construction lines into the foreground. The final animation consists of 101 individual frames.

```
\documentclass{article}
\usepackage{intcalc} %defines \intcalcMod for Modulo computation
\usepackage{animate}
\usepackage{graphics}

\newcounter{scarab}
\setcounter{scarab}{0}
\newcounter{blueline}
\setcounter{blueline}{101}
\newcounter{grayline}
\setcounter{grayline}{202}

%write timeline file
\newwrite\TimeLineFile
\immediate\openout\TimeLineFile=scarab.tln
```

```
\whiledo{\thescarab<101}{
  \ifthenelse{\intcalcMod{\thescarab}{2}=0}{
    %a gray line is added to every 2nd frame
    \immediate\write\TimeLineFile{%
      ::\thegrayline x0;253;\thescarab x0,\theblueline}
    \stepcounter{grayline}
  }{
    \immediate\write\TimeLineFile{%
      ::;253;\thescarab x0,\theblueline}
  }
  \stepcounter{scarab}
  \stepcounter{blueline}
}
\immediate\closeout\TimeLineFile

\begin{document}

\begin{center}
  \animategraphics[
    width=0.8\linewidth,
    controls, loop,
    timeline=scarab.tln
  ]{12}{scarab_}{0}{253}
\end{center}

\end{document}
```

7.2 Animating PSTricks graphics, using 'animateinline' environment

Fig. 3 is an inline graphics example adapted from [2].

```
\documentclass{article}
\usepackage{pst-3dplot}
\usepackage{animate}

%draws a torus sector
\newcommand{\torus}[2]{% #1: angle of the torus sector,
  %                      #2: linewidth of leading circle
  \psset{Beta=20,Alpha=50,linewidth=0.1pt,origin={0,0,0},unit=0.35}%
  \begin{pspicture}(-12.3,-6.3)(12.3,7)%
    \parametricplotThreeD[xPlotpoints=100](80,#1)(0,360){%
      t cos 2 mul 4 u sin 2 mul add mul
      t sin 2 mul 4 u sin 2 mul add mul
      u cos 4 mul
    }%
    \parametricplotThreeD[yPlotpoints=75](0,360)(80,#1){%
      u cos 2 mul 4 t sin 2 mul add mul
      u sin 2 mul 4 t sin 2 mul add mul
      t cos 4 mul
    }%
    \parametricplotThreeD[yPlotpoints=1,linewidth=#2](0,360)(#1,#1){%
      u cos 2 mul 4 t sin 2 mul add mul
```

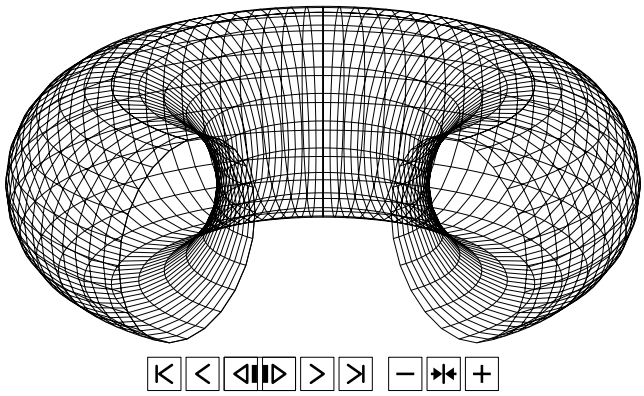

Figure 3

```
      u sin 2 mul 4 t sin 2 mul add mul
      t cos 4 mul
    }%
  \end{pspicture}%
}

\begin{document}

\begin{center}
\begin{animateinline}[poster=last, controls, palindrome]{12}%
  \multiframe{29}{iAngle=80+10, dLineWidth=2.9pt+-0.1pt}{%
    %iAngle = 80, 90, ..., 360 degrees
    %dLineWidth = 2.9pt, 2.8pt, ..., 0.1pt
    \torus{\iAngle}{\dLineWidth}%
  }%
\end{animateinline}%
\end{center}

\end{document}
```

8 Bugs

- The maximum frame rate that can actually be achieved largely depends on the complexity of the graphics and on the available hardware. Starting with version 8, Adobe Reader appears to be somewhat slower. However, you might want to experiment with the graphical hardware acceleration feature that was introduced in Reader 8. Go to menu 'Edit' → 'Preferences' → 'Page Display' → 'Rendering' to see whether hardware acceleration is available. A 2D GPU acceleration check box will be visible if a supported video card has been detected.
- The Adobe Reader setting 'Use page cache' (menu 'Edit' → 'Preferences' → 'Startup') should be *dis*abled for versions 6 & 7, while remaining

*en*abled beginning with version 8 (menu 'Edit' → 'Preferences' → 'Page Display' → 'Rendering').

- The `dvips` option '`-Ppdf`' should be avoided entirely or followed by something like '`-X 2400 -Y 2400`' on the command line in order to set a sensible DVI resolution. In times of Type-1 fonts, this does *not* degrade the output quality! The configuration file 'config.pdf' loaded by option '`-Ppdf`' specifies an excessively high DVI resolution that will be passed on to the final PDF. Eventually, Adobe Reader gets confused and will not display the frames within the animation widget.
- Animations do not work if the PDF has been produced with Ghostscript versions older than 8.31. This applies to all versions of ESP Ghostscript that comes with many Linux distributions.
- If the LaTeX → `dvips` → `ps2pdf`/Distiller route is being taken make sure that the original (unscaled) graphics size does not exceed the page size of the final document. During PS to PDF conversion every graphic of the animation is temporarily moved to the lower left page corner. Those parts of the graphics that do not fit onto the document page will be clipped in the resulting PDF. Fortunately, graphics files for building animations may be resized easily to fit into a given bounding box by means of the '`epsffit`' command line tool:

  ```
  epsffit -c <llx> <lly> <urx> <ury> infile.eps outfile.eps
  ```

  `<llx> <lly> <urx> <ury>` are the bounding box coordinates of the target document. They can be determined using Ghostscript. For a document named 'document.ps' the command line is

  ```
  gs -dNOPAUSE -q -dBATCH -sDEVICE=bbox document.ps
  ```

  Note that the name of the Ghostscript executable may vary between operating systems (e. g. '`gswin32c.exe`' on Win/DOS).
- Animations with complex graphics and/or many frames may cause LaTeX to fail with a '`TeX capacity exceeded`' error. The following steps should fix most of the memory related problems.

  MiKTeX:

  1. Open a DOS command prompt window (execute 'cmd.exe' via 'Start' → 'Run').
  2. At the DOS prompt, enter
     `initexmf --edit-config-file=latex`
  3. Type
     `main_memory=10000000`
     into the editor window that opens, save the file and quit the editor.
  4. To rebuild the format, enter
     `initexmf --dump=latex`
  5. Repeat steps 2–4 with config files '`pdflatex`' and '`xelatex`'

  TeX Live:

1. Find the configuration file 'texmf.cnf' by means of
   `kpsewhich texmf.cnf`
   at the shell prompt in a terminal.
2. As Root, open the file in your favourite text editor, scroll to the '`main_memory`' entry and change it to the value given above; save and quit.
3. Rebuild the formats by
   ```
   fmtutil-sys --byfmt latex
   fmtutil-sys --byfmt pdflatex
   fmtutil-sys --byfmt xelatex
   ```

- Animations should not be placed on *multilayered* slides created with presentation making classes such as Beamer or Powerdot. Although possible, the result might be disappointing. Put animations on flat slides only. (Of course, slides without animations may still have overlays.)

9 Acknowledgements

I would like to thank François Lafont who discovered quite a few bugs and made many suggestions that helped to improve the functionality of the package. Many thanks to Jin-Hwan Cho, the developer of '`dvipdfmx`', for contributing the '`dvipdfmx`' specific code.

References

[1] Chupin, M.: `http://melusine.eu.org/syracuse/metapost/animations/chupin/?idsec=scara`

[2] Gilg, J.: PDF-Animationen. In: *Die TEXnische Komödie*, Issue 4, 2005, pp. 30–37

[3] Holeček, J.: *Animations in a pdfTEX-generated PDF*. URL: `http://www.fi.muni.cz/~xholecek/tex/pdfanim.xhtml`

[4] *The Movie15 Package*. URL: `http://www.ctan.org/tex-archive/macros/latex/contrib/movie15`