The graphicscache package

Max Schwarz
max.schwarz@online.de

CTAN: http://www.ctan.org/pkg/graphicscache

v0.4 from 2022/12/20

1 Introduction

The `graphicx` package offers the versatile `\includegraphics` command, which offers image transformations like scaling, cropping, rotation, etc. However, these transformations have to be performed on every compilation of the document. Users can avoid this with the `draft` option at the cost of not seeing the images.

Furthermore, images are always included as-is with full resolution, even if they are shown at a very small actual size. This increases compilation time again and leads to large output files. A typical solution is to resize the input images to a lower resolution—but here, the user has to manually calculate or guess the required resolution. What we really want is to specify a *document DPI*, which automatically leads to the correct image resolution. This is possible using post-processing tools like ghostscript, but these do not help with compilation times and are typically not applicable for preprint servers or journals which require LaTeX sources.

`graphicscache` aims to solve these problems by decoupling the rendering of images from the actual inclusion. Images are rendered to the correct size using a separate `pdflatex` call, post-processed with ghostscript, and then included as PDF. As a bonus, the resulting PDF is cached, resulting in very fast recompilation times.

2 Usage

`graphicscache` requires the usage of the `\write18` call (also called shell escape). For `pdflatex`, you have to specify the `-shell-escape` argument during compilation. After enabling shell-escape, simply call

```
\usepackage{graphicscache}
```

to enable caching.

`graphicscache` overrides the `\includegraphics` command so that you can use it as usual. Internally, it calculates a hash from the `includegraphics` arguments and the package options to generate a cache key. If you change the input image file or the options, the file will be automatically rendered again.

Note: The first compilation process might take a while, since the cached PDFs are generated one-by-one.

2.1 Generated files

`graphicscache` will create a folder called `graphicscache` in the compilation directory. Additionally, latex output files under the jobname `graphicscacheout` will be created. All of these files and folders are temporary and can be deleted safely (at the cost of re-creating the cache).

2.2 Package options

`compress=false|flat|jpeg`

Specifies the image compression algorithm. If `false`, the ghostscript call is skipped, thus embedding the image at its original resolution and format. If `flat`, the lossless `FlatEncode` algorithm is chosen. Finally, `jpeg` indicates that JPEG encoding using `DCTEncode` is to be performed.

Note: In the `flat` case, the images are still downsampled to match the DPI specified using the `dpi` key.

`dpi=`⟨*number*⟩

Controls the image resolution in dots per inch. The default value is 300. This option only takes effect if `compress` is not `false`.

`qfactor=`⟨*number*⟩

This controls the quality parameter of the JPEG encoder (see `compress`). Smaller values give higher quality. The default value is 0.15, which corresponds to the "Maximum" setting mentioned by Adobe.

`listing=true|false`

If enabled, `graphicscache` will write an extra `.graphicscache` file with mappings from `includegraphics` arguments to cache files. This can be used to produce a version of the TeX source code that directly references the PDF files instead of the original sources.

`render=true|false`

Controls whether rendering is allowed. The default is `true`. If `false`, `graphicscache` is not allowed to create new cache files. Instead, it will attempt to use the appropriate cache file. If it does not exist, `graphicscache` will fall back to `graphicx` in-place rendering. This can be used to perform a final release (see Section 3.1).

`cachedir=`⟨*dir*⟩

This key can be used to move the cache directory to another location. The default value is `graphicscache`.

2.3 Macros

`\includegraphics[args]{path}`

This behaves exactly like the original `graphicx` `\includegraphics`. However, only a limited number of keys in `args` is supported at the moment: `width`, `height`, `trim`, `clip`, `angle`, `origin`, `keepaspectratio`, `scale`. In addition, you can specify any of the package options above here as well. For example, you might want to disable compression for a particular image with

`\includegraphics[width=...,compress=false]{...}`.

3 Tips & Tricks

3.1 Releasing your manuscript

In case you want to upload your manuscript to a journal or preprint server, you can use `graphicscache` to make your work easier:

1. Use `latexpand` to strip comments and flatten your tex sources into one file (optional).
2. Compile the file as usual to generate the cache files.
3. Compile the file again with the `-recorder` command line option. Here, we want to hide accesses of the original image files (`render=false`), but changing the package options will change the cache hash. For this purpose, `graphicspath` also reacts to a global definition of `\graphicscache@inhibit`, which has the same effect as `render=false`.
4. The generated `.fls` file will contain all cache files that are needed to compile your manuscript. Package them and your `.tex` file.

This process is automated in the `release.sh` script included in the distribution.

4 Implementation

```
\NeedsTeXFormat{LaTeX2e}[1994/06/01]
\ProvidesPackage{graphicscache}[2018/10/02 Graphics Cache]
\RequirePackage{graphicx}
\RequirePackage{xstring}
\RequirePackage{filemod}
\RequirePackage{letltxmacro}
\RequirePackage{pgfopts}
\RequirePackage{pgffor}
\RequirePackage{ifplatform}
\RequirePackage{pdftexcmds}
\RequirePackage{ltxcmds}
\newif\ifgraphicscache@render
\newif\ifgraphicscache@compress
\newif\ifgraphicscache@listing
\newif\ifgraphicscache@hashshortnames
\newif\ifgraphicscache@gsnotavailable\graphicscache@gsnotavailablefalse
\def\graphicscache@graphicsargs{}
\newlength\graphicscache@tmplen
\newcommand{\graphicscache@addarg}[1]{%
  \ifx\graphicscache@graphicsargs\empty
    \edef\graphicscache@graphicsargs{#1}%
  \else
    \edef\graphicscache@graphicsargs{\graphicscache@graphicsargs,#1}%
  \fi
}
\pgfkeys{
  /graphicscache/.cd,
  render/.is if=graphicscache@render,
  render=true,

  cachedir/.store in=\graphicscache@cachedir,
  cachedir={graphicscache},

  compress/.is choice,
  compress/false/.code={\graphicscache@compressfalse},
  compress/jpeg/.code={\graphicscache@compresstrue \def\graphicscache@compress@mode{DCTEncode}},
```

```
35  compress/flat/.code={\graphicscache@compresstrue ⟋
       \def\graphicscache@compress@mode{FlatEncode}},
36  compress=jpeg,

37  dpi/.store in=\graphicscache@dpi,
38  dpi=300,

39  qfactor/.store in=\graphicscache@qfactor,
40  qfactor={0.15},

41  hashshortnames/.is if=⟋
       graphicscache@hashshortnames,
42  hashshortnames=false,
```

We now define the list of supported graphicx arguments:

```
43  width/.code={%
44    \setlength\graphicscache@tmplen{#1}%
45    \graphicscache@addarg{width=\the\⟋
         graphicscache@tmplen}%
46  },
47  height/.code={%
48    \setlength\graphicscache@tmplen{#1}%
49    \graphicscache@addarg{height=\the\⟋
         graphicscache@tmplen}%
50  },
51  trim/.code={\graphicscache@addarg{trim=#1}},
52  clip/.code={\graphicscache@addarg{clip}},
53  angle/.code={%
54    \edef\graphicscache@tmp{#1}%
55    \graphicscache@addarg{angle=\graphicscache@tmp}⟋
         %
56  },
57  origin/.code={\graphicscache@addarg{origin=#1}},
58  keepaspectratio/.code={\graphicscache@addarg{⟋
       keepaspectratio}},
59  scale/.code={%
60    \edef\graphicscache@tmp{#1}%
61    \graphicscache@addarg{scale=\graphicscache@tmp}⟋
         %
62  },
63  page/.code={%
64    \edef\graphicscache@tmp{#1}%
65    \graphicscache@addarg{page=\graphicscache@tmp}%
66  },
```

```
  listing/.is if=graphicscache@listing,
  listing=false,
%
% adjustbox package
%
  frame/.code={%
    \edef\graphicscache@tmp{#1}%
    \graphicscache@addarg{frame=\graphicscache@tmp}%
  },
  valign/.code={%
    \edef\graphicscache@tmp{#1}%
    \graphicscache@addarg{valign=\graphicscache@tmp}%
  },
  raise/.code={%
    \edef\graphicscache@tmp{#1}%
    \graphicscache@addarg{raise=\graphicscache@tmp}%
  },
}
\ProcessPgfOptions{/graphicscache}\relax
\ifdefined\graphicscache@inhibit
  \pgfkeys{/graphicscache/render=false}%
\fi
\ifgraphicscache@listing
  \newwrite\graphicscache@listout
  \immediate\openout\graphicscache@listout=\jobname.graphicscache
\fi
%
% shellesc has a bug on Ubuntu 16.04 (\ShellEscape is not immediate).
% So we simply define our own shellescape macro.
\ifx\lastsavedimageresourcepages\@undefined
  \protected\def\graphicscache@ShellEscape{\immediate\write18 }
\else
  \protected\def\graphicscache@ShellEscape#1{%
    \directlua{os.execute("\luaescapestring{#1}")}}
\fi
```

\graphicscache@callgswithname

This macro calls ghostscript using the name specified in the first argument.

```
\newcommand{\graphicscache@callgswithname}[1]{%
  \ifwindows
    \graphicscache@ShellEscape{#1
      -sOutputFile=\graphicscache@output\space
      -sDEVICE=pdfwrite
      -dCompatibilityLevel=1.4
      -dPDFSETTINGS=/prepress
      -dNOPAUSE -dQUIET -dBATCH
      -c "<<
        /AutoFilterColorImages false
        /EncodeColorImages true
        /ColorImageFilter /\
           graphicscache@compress@mode\space
        /ColorImageDict << /ColorTransform 1 /
           QFactor \graphicscache@qfactor\space /
           Blend 1 /HSamples [1 1 1 1] /VSamples [1
            1 1 1] >>
        /ColorImageResolution \graphicscache@dpi\
           space
        /AutoFilterGrayImages false
        /EncodeGrayImages true
        /GrayImageFilter /\
           graphicscache@compress@mode\space
        /GrayImageDict << /ColorTransform 1 /
           QFactor \graphicscache@qfactor\space /
           Blend 1 /HSamples [1 1 1 1] /VSamples [1
            1 1 1] >>
        /GrayImageResolution \graphicscache@dpi\
           space
      >> setdistillerparams"
      -f \graphicscache@cachedir\string\
         graphicscacheout.pdf
    }%
  \else
    \graphicscache@ShellEscape{#1
      -sOutputFile=\graphicscache@output\space
      -sDEVICE=pdfwrite
      -dCompatibilityLevel=1.4
      -dPDFSETTINGS=/prepress
      -dNOPAUSE -dQUIET -dBATCH
```

```
131         -c '<<
132           /AutoFilterColorImages false
133           /EncodeColorImages true
134           /ColorImageFilter /\
                 graphicscache@compress@mode\space
135           /ColorImageDict << /ColorTransform 1 /
                 QFactor \graphicscache@qfactor\space /
                 Blend 1 /HSamples [1 1 1 1] /VSamples [1
                  1 1 1] >>
136           /ColorImageResolution \graphicscache@dpi\
                 space
137           /AutoFilterGrayImages false
138           /EncodeGrayImages true
139           /GrayImageFilter /\
                 graphicscache@compress@mode\space
140           /GrayImageDict << /ColorTransform 1 /
                 QFactor \graphicscache@qfactor\space /
                 Blend 1 /HSamples [1 1 1 1] /VSamples [1
                  1 1 1] >>
141           /GrayImageResolution \graphicscache@dpi\
                 space
142         >> setdistillerparams'
143         -f \graphicscache@cachedir/graphicscacheout.
               pdf \string|\string| rm \
               graphicscache@output
144       }%
145     \fi
146   }
```

\graphicscache@callgs

This macro finds the correct ghostscript executable to call.

```
147   \newcommand{\graphicscache@callgs}{%
```

If we previously established gs is not available, do nothing.

```
148     \ifgraphicscache@gsnotavailable
149     \else
150       \@ifundefined{graphicscache@gscommand}{%
151         \foreach \cmd in {rungs,gs,mgs} {%
152           \PackageInfo{graphicscache}{Trying \cmd\
                 space to call ghostscript...^^J}%
153           \graphicscache@callgswithname{\cmd}%
154           \IfFileExists{\graphicscache@output}{%
```

```
155             \PackageInfo{graphicscache}{Found a ⟋
                   working ghostscript called '\cmd'.}%
156             \global\edef\graphicscache@gscommand{\⟋
                   cmd}%
157             \breakforeach
158         }{}%
159       }%
160       \@ifundefined{graphicscache@gscommand}{%
161         \PackageWarning{graphicscache}{Could not ⟋
              find a working ghostscript executable. I⟋
               will not compress any images.}{}%
162         \graphicscache@gsnotavailabletrue
163       }{}%
164     }{%
165       \PackageInfo{graphicscache}{Calling gs with ⟋
            name '\graphicscache@gscommand'^^J}
166       \graphicscache@callgswithname{\⟋
            graphicscache@gscommand}
167     }%
168   \fi
169 }
```

\graphicscache@dorender

Here, we actually perform the rendering. Sadly, this is quite complex due to cross-platform support.

```
170 \newcommand{\graphicscache@dorender}{%
171   \PackageInfo{graphicscache}{Rendering \⟋
        graphicscache@outputhash: \graphicscache@fname⟋
        \space with args: \graphicscache@graphicsargs\⟋
        space (master file)}%
172   \ifwindows
173     \graphicscache@ShellEscape{md "\⟋
          graphicscache@cachedir" 2>NUL}%
174   \else
175     \graphicscache@ShellEscape{mkdir -p "\⟋
          graphicscache@cachedir"}%
176   \fi
```

First, render the graphics.

```
177   \ifwindows
178     \graphicscache@ShellEscape{del /q \⟋
          graphicscache@cachedir\string\⟋
```

```
      graphicscacheout.pdf}
    \graphicscache@ShellEscape{pdflatex
      -jobname graphicscacheout
      -interaction nonstopmode
      -output-directory "\graphicscache@cachedir"
      "\string\documentclass{standalone}
      \string\usepackage{graphicx}
      \string\usepackage[export]{adjustbox}
      \string\begin{document}\string\
        includegraphics[\
        graphicscache@graphicsargs]{\
        graphicscache@fname}\string\end{document}"
    }%
    \IfFileExists{\graphicscache@cachedir/
      graphicscacheout.pdf}{}{%
      \PackageWarning{graphicscache}{External
        pdflatex call failed (see above)}%
      \def\graphicscache@output{}%
    }
  \else
    \graphicscache@ShellEscape{pdflatex
      -jobname graphicscacheout
      -interaction nonstopmode
      -output-directory "\graphicscache@cachedir"
      '\string\documentclass{standalone}
      \string\usepackage{graphicx}
      \string\usepackage[export]{adjustbox}
      \string\begin{document}\string\
        includegraphics[\
        graphicscache@graphicsargs]{\
        graphicscache@fname}\string\end{document}'
      > /dev/null \string|\string| rm "\
        graphicscache@cachedir/graphicscacheout.
        pdf"
    }%
  \fi
```

Now, call ghostscript for compression—if required, otherwise just copy the file.

```
  \ifgraphicscache@compress
    \PackageInfo{graphicscache}{With compression: \
      graphicscache@compress@mode}%
    \graphicscache@callgs
  \else
```

```
    \PackageInfo{graphicscache}{Direct}%
    \ifwindows
      \graphicscache@ShellEscape{
        copy \graphicscache@cachedir\string\
           graphicscacheout.pdf \
           graphicscache@output
      }%
    \else
      \graphicscache@ShellEscape{
        cp \graphicscache@cachedir/graphicscacheout
           .pdf \graphicscache@output
      }%
    \fi
  \fi
}
```

save original \includegraphics

```
\LetLtxMacro\graphicscache@includegraphics\
   includegraphics%
\newcommand\graphicscache@native{%
  \expandafter\graphicscache@includegraphics\
     expandafter[\graphicscache@graphicsargs]{\
     graphicscache@fname}%
}
```

\graphicscache@work

This macro performs the update check: Do we need to render the file again or can we just include the cached version?

```
\newcommand{\graphicscache@work}{%
  \ifgraphicscache@render
```

Check if output file exists and is newer than input

```
    \filemodcmp{\graphicscache@fname}{\
       graphicscache@output}{% input is newer
      \graphicscache@dorender%
    }{% Output is newer
      \PackageInfo{graphicscache}{Already have \
         graphicscache@outputhash: \
         graphicscache@fname}%
    }%
```

If it still does not exist, we are likely in a strange environment (e.g. tabu). In that case, fall back to original includegraphics.

```
    \filemodcmp{\graphicscache@fname}{\
       graphicscache@output}{% input is newer/
       output does not exist
      \graphicscache@native
    }{% otherwise, use the generated file!
      \graphicscache@includegraphics{\
         graphicscache@output}%
    }%
  \else
```

Here, we just look if the output file exists. If not, fall back to original includegraphics.

```
    \IfFileExists{\graphicscache@output}{%
      \graphicscache@includegraphics{\
         graphicscache@output}%
    }{%
      \PackageWarning{graphicscache}{Could not find
          cache file \graphicscache@output, for \
         graphicscache@fname, falling back to
         native...}{}%
      \graphicscache@native
    }%
  \fi
}
```

\graphicscache@getfname

This macro resolves base names (i.e. includegraphics arguments with or without extensions) to relative paths.

```
\catcode`\*=11
\newif\ifgraphicscache@exists
\newcommand{\graphicscache@getfname}[1]{%
  \ifx\detokenize\@undefined\else
    \edef\Gin@extensions{\detokenize\expandafter{\
       Gin@extensions}}%
  \fi
  \begingroup
  \global\graphicscache@existstrue
  \let\input@path\Ginput@path
  \ltx@ifpackagelater{graphics}{2017/06/26}{%
    \set@curr@file{#1}%
    \expandafter\filename@parse\expandafter{\
       @curr@file}%
```

```
    \ifx\filename@ext\Gin@gzext
      \expandafter\filename@parse\expandafter{\
         filename@base}%
      \ifx\filename@ext\relax
        \let\filename@ext\Gin@gzext
      \else
        \edef\Gin@ext{\Gin@ext\Gin@sepdefault\
           Gin@gzext}%
      \fi
    \fi
  }{%
    \filename@parse{#1}%
  }%
  \ifx\filename@ext\relax
    \@for\Gin@temp:=\Gin@extensions\do{%
      \ifx\Gin@ext\relax
        \Gin@getbase\Gin@temp
      \fi}%
  \else
    \Gin@getbase{\Gin@sepdefault\filename@ext}%
    \ltx@ifpackagelater{graphics}{2017/06/26}{%
      \ifx\Gin@ext\relax
      \let\Gin@savedbase\filename@base
      \let\Gin@savedext\filename@ext
        \edef\filename@base{\filename@base\
           Gin@sepdefault\filename@ext}%
        \let\filename@ext\relax
         \@for\Gin@temp:=\Gin@extensions\do{%
            \ifx\Gin@ext\relax
              \Gin@getbase\Gin@temp
            \fi}%
        \ifx\Gin@ext\relax
          \let\filename@base\Gin@savedbase
          \let\filename@ext\Gin@savedext
        \fi
      \fi
    }{}%
  \fi
  \ifx\Gin@ext\relax
    \global\graphicscache@existsfalse
  \else
    \@ifundefined{Gin@rule@\Gin@ext}%
      {\global\graphicscache@existsfalse}%
      {}%
```

```
    \fi
    \ifgraphicscache@exists
      \xdef\graphicscache@fname{\Gin@base\Gin@ext}%
    \fi
    \endgroup
}
\catcode`\*=12
```

\includegraphics

Main entry point.

```
\renewcommand{\includegraphics}[2][]{%
    \begingroup
    \expandarg
```

Hash everything!

```
    \edef\graphicscache@options{\@nameuse{⟋
       opt@graphicscache.sty}}%
    \pgfkeys{/graphicscache/.cd,#1}%
```

If we are rendering, we need the actual filename, so that we can check modification times. Otherwise, just assume the file exists, `\includegraphics` will throw an error itself otherwise.

```
    \ifgraphicscache@render
      \graphicscache@getfname{#2}%
    \else
      \edef\graphicscache@fname{#2}%
      \graphicscache@existstrue
    \fi
    \ifgraphicscache@exists
      \ifgraphicscache@hashshortnames
        \edef\graphicscache@hashedname{#2}%
      \else
        \edef\graphicscache@hashedname{\⟋
           graphicscache@fname}%
      \fi
      \edef\graphicscache@outputhash{\pdf@mdfivesum{\⟋
         graphicscache@options\⟋
         graphicscache@graphicsargs\⟋
         graphicscache@hashedname}}%
      \edef\graphicscache@output{\⟋
         graphicscache@cachedir/\⟋
         graphicscache@outputhash.pdf}%
```

```
    \ifgraphicscache@listing
      \PackageInfo{graphicscache}{graphicscache: ⟋
         includegraphics\{#2\} => \⟋
         graphicscache@output}%
      \immediate\write\graphicscache@listout{#2 \⟋
         graphicscache@fname\space \⟋
         graphicscache@output}%
    \fi
    \graphicscache@work
  \else
    \PackageError{graphicscache}{Could not find ⟋
       file #2}{}%
  \fi
  \endgroup
}
```

\includegraphicscache

```
\newcommand{\includegraphicscache}[3][]{%
  \begingroup
  \expandarg
  \pgfkeys{/graphicscache/.cd,#2}%
  \includegraphics[#1]{#3}%
  \endgroup
}

\endinput
```