draw pixel pictures

Jonathan P. Spratte

2021-12-12 v1.3

Abstract
With \texttt{pxpic} you draw pictures pixel by pixel. It was inspired by a lovely post by Paulo Cereda, among other things (most notably a beautiful duck) showcasing the use of characters from the Mario video games by Nintendo in \LaTeX.

Contents

1 Documentation
  1.1 Drawing pictures ......................................................... 2  
    1.1.1 Examples ................................................................ 2  
  1.2 Setting options .............................................................. 4  
    1.2.1 Colour syntax .......................................................... 5  
    1.2.2 Available modes ...................................................... 5  
  1.3 Other customisation macros .............................................. 6  
  1.4 Other macros .................................................................. 6  
  1.5 Miscellaneous ............................................................. 7  

2 Implementation
  2.1 Options .................................................................. 9  
  2.2 User macros ................................................................. 10  
  2.3 Parser .................................................................. 13  
  2.4 Modes .................................................................. 14  
  2.5 Pixel and Skip ............................................................. 15  
  2.6 Parser for colours ......................................................... 15  
  2.7 Messages ................................................................. 16

Index

\texttt{jspratte@yahoo.de}
1 Documentation

1.1 Drawing pictures

\texttt{\textbackslash pxpic} supports different input modes, all of them have the same basic parsing behaviour. A \(\langle \text{pixel list} \rangle\) contains the pixel colours. The image is built line wise from top left to bottom right. Each row of pixels should be a single \TeX{} argument (so either just one token, or a group delimited by \{\}), and within each line each pixel in turn should be a single \TeX{} argument (so either just one token, or a group delimited by \{\}). Spaces and hence single newlines in the sources between \(\langle \text{pixel list} \rangle\) elements are ignored. The different modes are explained in \texttt{subsection 1.2.2}. The only disallowed token in the \(\langle \text{pixel list} \rangle\) is the control sequence \texttt{\textbackslash pxpic\textbackslash end} (plus the usual restrictions of \TeX{} so no unbalanced braces, no macros defined as \texttt{\textbackslash outer}).

There is a small caveat however: \texttt{\textbackslash pxpic} draws each pixel individually, and there is really no space between them, however some \texttt{pdf} viewers fail to display such adjacent lines correctly and leave small gaps (basically the same issue which packages like \texttt{colortbl} suffer from as well). In print this shouldn’t be an issue, but some rasterisation algorithms employed by viewers and conversion tools have this deficit.

Another thing I should mention: The pictures you can draw with \texttt{\textbackslash pxpic} can’t be arbitrary large. Due to the design decision of the output as a single \\texttt{\hbox} and the way the output routine works, pictures are limited by \TeX{}’s memory size to roughly \(440 \times 440\) pixels in \texttt{pdf\LaTEX} with the default settings in \TeX{} Live. The size is unlimited in \texttt{Lua\LaTEX}, due to dynamic memory allocation. In \texttt{Xe\LaTEX} the size should be even smaller than in \texttt{pdf\LaTEX}.

\begin{verbatim}
\pxpic \pxpic[(\textit{options})](\langle \text{pixel list} \rangle)
\end{verbatim}

\(\langle \text{options} \rangle\) might be any options as listed in \texttt{subsection 1.2}, and \(\langle \text{pixel list} \rangle\) is a list of pixels as described above. \texttt{\textbackslash pxpic} parses the \(\langle \text{pixel list} \rangle\) and draws the corresponding picture. The result is contained in an \\texttt{\hbox} and can be used wherever \TeX{} expects an \\texttt{\hbox}. As a result, when you’re in vertical mode a \texttt{\textbackslash pxpic} will form a text line, to prevent this you can use \texttt{\textbackslash leavevmode} before it. The \texttt{\textbackslash pxpic} will be bottom aligned by default (see the options \texttt{b}, \texttt{c}, and \texttt{t}), you can further tweak this using \texttt{\textbackslash raisebox} (or, if you want, \TeX{}’s \texttt{\textbackslash raise} and \texttt{\textbackslash lower} primitives).

1.1.1 Examples

Since the above explanation of the \(\langle \text{pixel list} \rangle\) syntax might’ve been a bit cryptic, and a good documentation should contain examples (this doesn’t claim this documentation is good), well, here are some examples (you might need to take a look at \texttt{subsection 1.2} and \texttt{subsubsection 1.2.2} to fully understand the examples). Examples in this section will use the following \texttt{\textbackslash pxpicsetup}:

\begin{verbatim}
\pxpicsetup
//
\end{verbatim}

\begin{verbatim}
  \mode = px
  \c\ol\ours = \{\texttt{k=black}, \texttt{r=[HTML\{9F393D\}}, \texttt{g=green!75!black}, \texttt{b=[rgb\{0,0,1\}}
  \skip = .
  \size = 10pt
\end{verbatim}
We can draw a small cross rather easily:

```latex
\pxpic
\{.k\}
\{kk\}
\{.k\}

A small multicoloured grid:

```latex
\pxpic
\{bgk\}
\{kbg\}
\{gkb\}
\{rgb\}

A heart (shamelessly copied example from \texttt{PixelArt}):  

```latex
\pxpic
\{...
..rr,.rr
{.rrrrrrrr}
{rrrrrrrrrr}
{rrrrrrrrrr}
{.rrrrrrrr}
{...rrr}
{....r}
\}

Using \texttt{mode=rgb} to draw a short coloured line:

```latex
\pxpic[mode=rgb]\{(1,0,1)(1,1,0)(0,1,1)\}

A multicoloured grid using skips and \texttt{mode=cmy}:

```latex
\pxpic[mode=cmy]
\{(1,0,1)(1,1,0)(0,1,1)\}
\{(1,0,1)(1,1,0)(0,1,1)\}
\{(0,1,1)(1,0,1)(1,1,0)\}
\{(1,1,0)(0,1,1)(1,0,1)\}

Showing the difference between a skipped and a white pixel:

```latex
\pxpicsetup{colours = \texttt{w=white}}
\colorbox{gray}{\pxpic{bbb}{b.b}{bbb}}
\colorbox{gray}{\pxpic{bbb}{bwb}{bbb}}

A biggish example: Tux.\footnote{Source: \url{https://www.reddit.com/r/linux/comments/hwp9ij/tux_pixel_art_v10/}} I put two rows of pixels per code line to reduce the size a
bit and the code is displayed tinily.

\[\text{Just for Paulo, a duck. Also, showing that the colour definitions in mode=px can be arbitrary tokens or multiple letters:}\]

Another example might be the definition of \texttt{\pxpiclogo} in subsection 2.2. Who still needs \texttt{picture}-mode or complicated packages like \texttt{pstricks} or \texttt{TikZ} with such pretty pictures?

\section*{1.2 Setting options}

To control its behaviour \texttt{\pxpic} uses a \texttt{key=value} interface powered by \texttt{expkv}. Options can be set either in the optional argument of \texttt{\pxpic} or with

\texttt{\pxpicsetup(\texttt{options})}

Sets the \texttt{(options)} locally to the current \TeX{} group.

Package options are not supported.

The available options are

\texttt{colors=(\texttt{colour list})}

Define pixel colours for \texttt{mode=px}, see \subsubsection*{1.2.1} for a description of the value's syntax. No pixel definitions are made by the package.

\texttt{colours} \texttt{see colours.}
color-list={(choice)}
loads a previously through \pxpicnewcolorlist defined colour list. No colour lists are
defined by the package.

colour-list see color-list.

gap-hack={(dimen)}
To fix the issues with visible gaps in PDF viewers you can introduce some negative kerns
to make the pixels overlap (lines overlap to the top, pixels to the left). This option expects
a dimension as its value. A positive value will (maybe) close the gaps, a negative value
will introduce real gaps. In any case the outermost pixels’ borders still coincide with
the borders of the surrounding \hbox. Take a look at my babbling about this issue in
subsection 1.5.

ht=(dimen) Set the height of the pixels.

mode={(choice)}
Set the used mode, see subsubsection 1.2.2 for available modes. Initial value is px.

size=(dimen)
Set both ht and wd. Initial value is 1.0pt.

skip={tokens}
Define \langle tokens \rangle to be a skip (an empty space of width wd) in mode=px. No skip definitions
are made by the package.

wd=(dimen) Set the width of the pixels.

b Set the bottom of the \pxpic on the surrounding baseline (vertical bottom alignment;
this is the default).

c Set the centre of the \pxpic on the surrounding baseline (vertical centre alignment).

t Set the top of the \pxpic on the surrounding baseline (vertical top alignment).

1.2.1 Colour syntax
In the value of the colours option you’ll have to use the following syntax. Use a comma
separated key=value list in which each key corresponds to a new pixel name for mode=px,
and each value to the used colour. If the colour starts with an opening bracket use the
complete value as is behind \color, else use the whole value as the first mandatory
argument to \color with a set of braces added. For example to define r as the named
colour red, and x as the colour #abab0f (in the HTML colour model) use:

\texttt{colours} = \texttt{\{r=red, x=[H\text{T}M\text{L}\{abab0f\}\}}

1.2.2 Available modes
px
As already mentioned, \pxpic supports different modes of input. The easiest to use mode
is px, in which each element of the \langle pixel list \rangle has been previously defined as either a
coloured pixel (using the colour option) or as a skipped pixel (using the skip option,
resulting in a fully transparent pixel). Each element will be \detokenized, so (within \TeX’s limitations) the name of a pixel can be arbitrary. This is the initial mode \pxpic uses. But other options are available as well.
Another mode is named, in which each element of the \langle pixel list \rangle should be a named colour (or colour expression) known to xcolor. Each element will be used like so: \{\color{\langle element \rangle}\px\}. An exception is an element which is empty (\{}), which will be a skipped pixel.

rgb, cmy, cmyk, hsb, Hsb, tHsb, gray, RGB, HTML, HSB, Gray, wave

The modes rgb, cmy, cmyk, hsb, Hsb, tHsb, gray, RGB, HTML, HSB, Gray, and wave correspond to the different colour models supported by xcolor. With these modes each element of the \langle pixel list \rangle will be the values in these colour models, so they'll be used like so: \{\color{\langle mode \rangle}\langle element \rangle\px\}. An exception is an element which is empty (\{}), which will be a skipped pixel.

You can define additional modes selectable with the mode option using the macros \pxpicnewmode or \pxpicsetmode.

1.3 Other customisation macros

\pxpicnewmode \pxpicsetmode

You can define your own modes with \pxpicnewmode. Inside \langle definition \rangle #1 is the currently parsed item in the \pxpic \langle pixel list \rangle. You can output a pixel using \px, and skip a pixel using \pxskip. The pixel will use the currently active colour (so if you want to draw a red pixel you could use \color{red}\px). \pxpicnewmode will throw an error if you try to define a mode which already exists, \pxpicsetmode has no checks on the name.

\pxpicnewcolorlist \pxpicsetcolorlist \pxpicaddcolorlist

This defines a colour list (to be used with the colour-list option). The syntax of \langle colour list \rangle is the same as for the colours option. The pixels aren't directly defined, but only by the use of colour-list=\langle name \rangle. So

\pxpicnewcolorlist{example}{r=red,b=blue,g=green,k=black,w=white}
\pxpicsetup{colour=example}

would have the same effect as

\pxpicsetup{colours={r=red,b=blue,g=green,k=black,w=white}}

but a colour-list is more efficient if used multiple times. The new variant will only throw an error if the colour list \langle name \rangle is already defined. The set variant has no such tests, and the add variant will add additional colours to an existing list.

\pxpicforget

Undefines the \langle px \rangle definition for use in mode=px (or skip symbol) added with the colours (or skip) option.

1.4 Other macros

\px \pxskip

Inside of a \pxpic the macro \px draws a pixel (of the currently active colour), and \pxskip leaves out a pixel (so this one pixel is fully transparent). Use this in the \langle definition \rangle of a mode in \pxpicnewmode.
These two are \texttt{dimen} registers storing the height and width of the pixels.

\texttt{\pxpicHT}
\texttt{\pxpicWD}

\texttt{\pxpiclogo\{\textit{size}\}}

This draws the logo of \texttt{pxpic}. The \textit{size} controls the pixel size.

1.5 Miscellaneous

If you find bugs or have suggestions I’ll be glad to hear about it, you can either open a ticket on Github (\url{https://github.com/Skillmon/ltx_pxpic}) or email me (see the first page).

A similar package is \texttt{PixelArt}, which, as of writing this, is described as a “working draft” by its author. \texttt{pxpic} wasn’t intended as a direct competitor (I already started coding when I learned about \texttt{PixelArt}’s existence), but I took inspiration from the “Bugs, Ideas, Undefined behaviours” section of \texttt{PixelArt}’s documentation for the syntax of \texttt{mode=px}.

Regarding the gap issue: The pixels are output touching each other with no real gap, however some PDF viewers and tools will display such a gap. To make things even worse, the effect depends on the viewers current magnification. \texttt{pxpic} has the \texttt{gap-hack} option to provide some crude hack that might fix the issue, at the cost that the pixels on the far right and bottom are bigger than they were specified to be. Also pixels next to skipped pixels have a different size (skipped pixels don’t cover pixels to their left or top as they are transparent). You’ll want to find a good trade-off value if you want to use \texttt{gap-hack}, that mitigates the effect but isn’t too big (to make the errors less obvious). You can play with the value and decide for yourself what’s the lesser evil. Or you do like me, don’t use \texttt{gap-hack} and blame the viewers. Here are examples in which you can compare (the \texttt{gap-hack} is chosen way too big in this example and skips are used close to white pixels on purpose, but it illustrates the effects; the third output, not shown in the code, uses a
more reasonable gap-hack=.2pt):
\pixmapsetup
  
  \ncolours={k=black,g=green,w=white}
  \nskip=
  \nsize=10pt
  \n\pixmap{gap-hack=2pt}
  \n  {kkkkk}
  {kgggk}
  {kgg.k}
  {kg.gk}
  {kgw}
  {kww}
  \n\pixmap
  \n  {kkkkk}
  {kgggk}
  {kgg.k}
  {kg.gk}
  {kgw}
  {kww}
2 Implementation

Report who we are

\ProvidesPackage{pxpic}[2021-12-12 v1.3 draw pixel pictures]
and load dependencies

\RequirePackage{xcolor}
\RequirePackage{expkv}

These two variables store the height and width of a pixel.

\@ifdefinable\pxpicHT{\newdimen\pxpicHT}
\@ifdefinable\pxpicWD{\newdimen\pxpicWD}
\pxpicHT=\p@
\pxpicWD=\pxpicHT

(End definition for \pxpicHT and \pxpicWD. These variables are documented on page 7.)

\pxpic@kern

To fix the visible gaps in some PDF viewers if the user chooses so with the \texttt{gap-hack} option we introduce some \texttt{@kerns} of the length stored in this register.

\@ifdefinable\pxpic@kern{\newdimen\pxpic@kern}
\pxpic@kern=\z@

(End definition for \pxpic@kern.)

\pxpic@inner@box \pxpic@after@inner@box

To get different vertical alignments we nest one of \texttt{vbox}, \texttt{vtop}, and a lowered \texttt{vbox} inside the outer \texttt{hbox}. The macro \texttt{\pxpic@inner@box} will store this information, and since lowering can only be done after the box was set (the alternative would be \texttt{vcenter}, which ends up on a different height), we need to be able to put the box output with \texttt{lower} after the box was collected, which is why we need \texttt{\pxpic@after@inner@box}. We default to bottom alignment.

\@ifdefinable\pxpic@inner@box{\let\pxpic@inner@box\vbox}
\@ifdefinable\pxpic@after@inner@box{\let\pxpic@after@inner@box\@empty}

(End definition for \pxpic@inner@box and \pxpic@after@inner@box)

2.1 Options

We define the options using \texttt{expkv} directly (no fancy options are involved and these are just a few anyway).

The first few options are straight forward. We use \texttt{expkv’s} name space to actually store the \texttt{skip} and \texttt{px} definitions, hence we use \texttt{\ekvdefNoVal} in the code of \texttt{skip}.

\@protected\ekvdef{\pxpic}{size}
\{\pxpicHT=\dimexpr\#1\relax\pxpicWD=\pxpicHT\}
\@protected\ekvdef{\pxpic}{ht}
\{\pxpicHT=\dimexpr\#1\relax\}
\@protected\ekvdef{\pxpic}{wd}
\{\pxpicWD=\dimexpr\#1\relax\}
\@protected\ekvdef{\pxpic}{gap-hack}
\{\pxpic@kern=\dimexpr\#1\relax\}
\@protected\ekvdef{\pxpic}{skip}
\{\ekvdefNoVal{\pxpic@px}{\#1}{\pxskip}\}

The \texttt{colours} option is parsed using \texttt{\ekvpars} and \texttt{\pxpic@setcolor}.

\@protected\ekvdef{\pxpic}{colors}
\{\ekvpars\pxpic@err@noval\pxpic@setcolor{\#1}\}
\ekvletkv{\pxpic}{colours}{\pxpic}{\pxpic}{\colors}
And the \texttt{mode} just checks whether the \texttt{mode} macro is defined and lets the auxiliary macro \texttt{\pxpic@parse@px} to the defined \texttt{mode}.

\begin{verbatim}
\protected\ekvdef{pxpic}{mode}
  {%
    \@ifundefined{pxpic@parse@px@#1}{%
    \pxpic@err@unknown@mode{#1}}%
    {% \expandafter\let\expandafter\pxpic@parse@px
      \csname pxpic@parse@px@#1\endcsname
    }%
  }
\end{verbatim}

A similar check is done for the \texttt{colour-list} option.

\begin{verbatim}
\protected\ekvdef{pxpic}{color-list}
  {% \@ifundefined{pxpic@colorlist@#1}{\pxpic@err@unknown@colorlist{#1}}
    \csname pxpic@colorlist@#1\endcsname
  }
\ekvletkv{pxpic}{colour-list}{pxpic}{color-list}
\end{verbatim}

The alignment options set the internals \texttt{\pxpic@inner@box} and \texttt{\pxpic@after@inner@box}.

\begin{verbatim}
\protected\ekvdefNoVal{pxpic}{b}
  {% \let\pxpic@inner@box\vbox
    \let\pxpic@after@inner@box\@empty
  }
\protected\ekvdefNoVal{pxpic}{c}
  {% \def\pxpic@inner@box{\setbox0=\vbox}%
    \def\pxpic@after@inner@box{\lower.5\ht0\box0}%
  }
\protected\ekvdefNoVal{pxpic}{t}
  {% \let\pxpic@inner@box\vtop
    \let\pxpic@after@inner@box\@empty
  }
\end{verbatim}

### 2.2 User macros

\texttt{\pxpic} expands directly to an opened \texttt{\hbox}, the auxiliary \texttt{\pxpic@} checks for the optional argument and inserts the rest of the code. We need to set \texttt{\baselineskip} to \texttt{\pxpicHT} so that the pixels are stacked vertically without gaps. \texttt{\pxpic@parse} will parse the \texttt{⟨pixel list⟩} until \texttt{\pxpic@end} is hit. The \texttt{\egroup} closes the \texttt{\hbox}. The row-wise output is done via a \texttt{\vbox} in which each pixel row will be wrapped inside an \texttt{\hbox}. The \texttt{\kern} negates a negative \texttt{\kern} in \texttt{\pxpic@parse} so that the first line isn’t moved.

\begin{verbatim}
\@ifdefinable\pxpic{\protected\def\pxpic{\hbox\bgroup\pxpic@}}
\newcommand\pxpic@[2][]
  {% \pxpicsetup[#1]% \pxpic@inner@box
    \% \let\px\pxpic@px
\end{verbatim}
\let\pxskip\pxpic@skip
\advance\pxpicHT\pxpic@kern
\advance\pxpicWD\pxpic@kern
\baselineskip=\pxpicHT
\kern\pxpic@kern
\pxpic@parse#2\pxpic@end
\%
\pxpic@after@inner@box
\egroup
(End definition for \pxpic and \pxpic@. These functions are documented on page 2.)

\pxpicsetup
Just directly defined to call expv’s parser for the \pxpic set.
\ektvsetdef\pxpicsetup{pxpic}
(End definition for \pxpicsetup. This function is documented on page 4.)

\pxpiclogo
The logo is just a biggish pixel picture. The \lower will move it down a bit so that it appears correctly aligned on the baseline. Since the logo should be part of a normal sentence in most usages we put \leavevmode before it. Also we make sure that the mode and px definitions are correct and the output is bottom aligned.
\newcommand*{\pxpiclogo}[1][.13ex]
{%
\begingroup
\pxpicHT=\dimexpr#1\relax
\pxpicWD=\pxpicHT
\pxpic@kern=\z@
\leavevmode
\lower3.2\pxpicHT\pxpic
[b, mode=px, colours={o=[HTML]{9F393D}, g=black!75},skip=.]
{............................................g}
{...........................................gggg}
{.oooo.......................gggg...........ggg}
{.ooooo...oo......oo...oo....ggggg...gg......g}
{.ooooooooooo...ooooo..oooo..gggggggggg...ggggg...ggggggg}
{..ooooo..oooo.ooooooooooo....ggggg..gggg.ggggggg.ggggggggg}
{...oooo..oooo.....oooo........gggg..gggg...gggg..gggg}
{...oooo..oooo.....oooo........gggg..gggg...gggg..gggg}
{.oooooo..oooo.....ooooo.....gggggg..gggg...gggg..gggg}
{oooooooooooo...ooooooooo...gggggggggg....gggg..ggggggggg}
{o.oooooooo....ooooo.oooooo.g.gggggggg......ggggg..ggggggg}
{...ooo.o......o.oo...oo.......ggg.g.........gg......ggg}
{...ooo........................ggg}
{...ooo........................ggg}
{....o..........................g}
\}%
\endgroup
(End definition for \pxpiclogo. This function is documented on page 7.)
\pxpicforget

Straight forward, just let the px macro to an undefined macro.
\newcommand{\pxpicforget[1]}
{\expandafter{\let\csname\ekv@name{pxpic@px}{#1}\endcsname\pxpic@undef}}

(End definition for \pxpicforget. This function is documented on page 6.)

\pxpicnewmode \pxpicsetmode

These are pretty simple as well, the new variant will use \newcommand which will do the testing for us, the set variant uses \def.
\protected\long\def{\pxpicnewmode[#1]}
{\expandafter{\newcommand\csname pxpic@parse@px@#1\endcsname[1]{#2}}}
\protected\long\def{\pxpicsetmode[#1]}
{\long\expandafter{\def\csname pxpic@parse@px@#1\endcsname\@gobble}}

(End definition for \pxpicnewmode and \pxpicsetmode. These functions are documented on page 6.)

\pxpicnewcolorlist \pxpicsetcolorlist \pxpicaddcolorlist \pxpic@setcolorlist \pxpic@addcolorlist

The colour list is first parsed with \ekvparse inside an \edef. \ekvparse will prevent the parsed list from further expanding, leaving each list element and \pxpic@experr@noval or \pxpic@setcolor@colorlist before it. In a second \edef these will be expanded, \pxpic@experr@noval throwing an error for each element missing a colour definition, and \pxpic@setcolor@colorlist testing for an opening bracket (which we do expandably) and leaving the correct definition protected against further expansion. The add variant uses a temporary macro for the parsing part and adds the result to the list holding macro. The second expansion step in set and both in add are done inside a group to revert any definition (also those letting tokens to \relax by \csname) made at this point except for the list macro itself.
\protected\def{\pxpicnewcolorlist[#1]}
{\@ifundefined{pxpic@colorlist@#1}
{\{\pxpicsetcolorlist[#1]\}}
{\pxpic@err@defined@colorlist{#1}\@gobble}\}}
\protected\def{\pxpicsetcolorlist[#1]}
{\expandafter{\pxpic@setcolorlist\csname pxpic@colorlist@#1\endcsname}}
\protected\long\def{\pxpic@setcolorlist[#1]}
{\begingroup\edef\pxpic@tmp\{\ekvparse\pxpic@experr@noval\pxpic@setcolor@colorlist{#1}\endgroup}}
\begingroup\edef\pxpic@tmp\{\endgroup\protected\def\unexpanded{\@gobble}\endgroup\@gobble}{#1}\@gobble}\}}
\protected\def{\pxpicaddcolorlist[#1]}
{\@ifundefined{pxpic@colorlist@#1}
{\pxpic@err@unknown@colorlist{#1}\@gobble}
{\expandafter{\pxpic@addcolorlist\csname pxpic@colorlist@#1\endcsname}}}
These are three helper macros. The first just gobbles everything until the next `\pxpic@end`, and we borrow a fast test for an empty argument from `expandafter`. The last can be used to check for an opening bracket if used like `\pxpic@ifbracket\pxpic@end #1.\pxpic@end[#2]\pxpic@end`. (End definition for `\pxpic@ifend`, `\pxpic@ifempty`, and `\pxpic@ifbracket`.)

For some weirder \TeX programming it is sometimes necessary to insert an unmatched opening brace. This code does exactly that if it’s expanded twice. It is put into a single macro so that one can `\expandafter` it easier.

The parsing loop is pretty simple, first check whether we’re done, else open a new `\hbox` (which will form a row in the `\vbox` placed by `\pxpic@`) in which the inner parsing loop is run. Then call the next iteration. If we’re done just gobble the remainder of the current iteration. First we introduce our `\kern` which might fix the gap issue. Another `\kern` is done at the start of each `\hbox` to compensate the unnecessary `\kern` done by the first `\pxpic@parseline`.

The line parsing loop also checks whether we’re done, if not we place a pixel using the current definition of `\pxpic@px` (which will be set by the current `mode`) and afterwards call the next iteration. If we’re done we gobble the remainder of the current iteration and control goes back to `\pxpic@parse`. Before each pixel we introduce a negative `\kern` to maybe fix the gap issue by letting the pixels overlap a bit.

(End definition for `\pxpic@parse` and `\pxpic@done`.)
\begin{verbatim}
\def\pxpic@parseline
\long\def\pxpic@linedone
\pxpic@end\kern-\pxpic@kern\pxpic@parse@px#1\pxpic@parseline
\}
\end{verbatim}

(End definition for \texttt{\pxpic@parseline} and \texttt{\pxpic@linedone}.)

2.4 Modes

The modes define how a single element of the \langle\texttt{pixel list}\rangle is parsed.

\texttt{\pxpic@parse@px@px@px} In the \texttt{px} mode we check whether the pixel is defined (using the name space of \texttt{exPkv}), if so call it, else throw an error and skip. Since this is also the initial mode we \texttt{\let} the auxiliary macro \texttt{\pxpic@parse@px} to this mode here.

\begin{verbatim}
\newcommand\pxpic@parse@px@px[1]{% 
  \ekvifdefinedNoVal{pxpic@px}{#1}{%\csname\ekv@name{pxpic@px}{#1}N\endcsname}% 
  \pxpic@err@unknown@px{#1} %
  \pxskip %
}\let\pxpic@parse@px\pxpic@parse@px@px
\end{verbatim}

(End definition for \texttt{\pxpic@parse@px@px@px} and \texttt{\pxpic@parse@px}.)

\texttt{\pxpic@parse@px@named} named just checks whether the skip is empty. If so skip, else call \texttt{\color} with the element and output a pixel.

\begin{verbatim}
\newcommand\pxpic@parse@px@named[1]{% 
  \pxpic@ifempty{#1}{\pxskip}{{\@declaredcolor{#1}\px}}%
}\end{verbatim}

(End definition for \texttt{\pxpic@parse@px@named}.)

\texttt{\pxpic@parse@px@rgb} \texttt{\pxpic@parse@px@cmy} \texttt{\pxpic@parse@px@cmyk} \texttt{\pxpic@parse@px@rgb} \texttt{\pxpic@parse@px@hsb} \texttt{\pxpic@parse@px@gray} \texttt{\pxpic@parse@px@hsb} \texttt{\pxpic@parse@px@HTML} \texttt{\pxpic@parse@px@RGB} \texttt{\pxpic@parse@px@Hsb} \texttt{\pxpic@parse@px@Gray} \texttt{\pxpic@parse@px@wave}

The colour model modes are all the same in principle. They test for an empty element to introduce a skip, else they call \texttt{\color} with the respective colour model and output a pixel. We use the auxiliary \texttt{\pxpic@tmp} to do all those definitions and undefine it afterwards.

\begin{verbatim}
\def\pxpic@tmp#1{% 
  \pxpic@newmode{#1}% 
  \%\pxpic@parse@px@#1% 
  \%\pxpic@parse@px@HSB% 
  \%\pxpic@parse@px@HTML% 
  \%\pxpic@parse@px@RGB% 
  \%\pxpic@parse@px@Hsb% 
  \%\pxpic@parse@px@Gray% 
  \%\pxpic@parse@px@wave% 
  \pxpic@tmp{rgb}% 
  \pxpic@tmp{cmy}% 
}\end{verbatim}

14
2.5 Pixel and Skip

The actual definition of pixels and skips is stored in macros to which the frontend macros \px and \pskip will be let inside of \pxpic.

\newcommand\pxpic@px\vrule height\pxpicHT width\pxpicWD depth\z@\endcsname
\newcommand\pxpic@skip\kern\pxpicWD\endcsname

(End definition for \pxpic@px and \pxpic@skip.)

2.6 Parser for colours

First we test whether the colour starts with an opening bracket or not. Depending on that we either just put the colour after \color, or put braces around it (as it then is a colour expression for \xcolor and just a single argument). \pxpic@setcolor defines a px in the name space of exPkv (this has a slight overhead during definition, but exPkv is fast in checking whether one of its keys is defined or not, and reduces the amount of code in this package).

\newcommand\pxpic@setcolor[2]{%\pxpic@ifbracket\pxpic@end#2.\pxpic@end\{#1\}{#2}%
\expandafter\def\csname\ekv@name{pxpic@px}{#1}N\endcsname
\@declaredcolor{#2}\px}}%
\newcommand\pxpic@setcolor@b[2]{%\expandafter\def\csname\ekv@name{pxpic@px}{#1}N\endcsname
\@undeclaredcolor#2\px}}%

(End definition for \pxpic@setcolor, \pxpic@setcolor@a, and \pxpic@setcolor@b.)

This macro should leave the correct code in the input stream to define a single pixel. It is to be used inside of \edef, hence using \unexpanded, which doesn’t have an opening brace directly after it so that the \pxpic@ifbracket test is fully expanded. Next we
expand \pxpic@setcolor@colorlist twice (which will expand the \csname contained in it)
and then leave the opening bracket for \unexpanded in the input stream. The code
should be used inside a group so that all the implicit definitions to \relax done by
\csname are reverted.

\newcommand\pxpic@setcolor@colorlist[2]
{%
\unexpanded\iffalse{\fi
\pxpic@ifbracket\pxpic@end#2.\pxpic@end\[]\pxpic@end
{\expandafter\expandafter\expandafter\pxpic@openbrace\pxpic@setcolor@a}
{\expandafter\expandafter\expandafter\pxpic@openbrace\pxpic@setcolor@b}
{#1}{#2}%
}%
}%

(End definition for \pxpic@setcolor@colorlist.)

2.7 Messages

These are just some macros throwing errors, nothing special here.

\newcommand\pxpic@err@noval[1]
{%\PackageError{pxpic}{Missing colour definition for name `\detokenize{#1}`}{}%}
\newcommand\pxpic@err@unknown@px[1]
{%\PackageError{pxpic}{Unknown pixel \'\detokenize{#1}'. Skipping}{}%}
\newcommand\pxpic@err@unknown@mode[1]
{%\PackageError{pxpic}{Unknown mode '#1'}{}%}
\newcommand\pxpic@err@unknown@colorlist[1]
{%\PackageError{pxpic}{Unknown colour list '#1'}{}%}
\newcommand\pxpic@err@defined@colorlist[1]
{%\PackageError{pxpic}{Colour list '#1' already defined}{}}

(End definition for \pxpic@err@noval and others.)

\pxpic@experr This macro can be used to throw an error expandably. For this an undefined control
sequence \pxpic␣Error: is used. The group containing \expandafter keeps the definition
of \pxpic␣Error: local (it is \relax after the \csname) so that it is undefined when
it’s used. The \@firstofone is needed to get the readable output (now the undefined
macro and actual message are always the same argument).

\def\pxpic@experr#1%
{%
\long\def\pxpic@experr##1%
{%\expandafter\expandafter\expandafter
\pxpic@ifend
\@firstofone{#1##1.}%
\pxpic@end%
}
}
\begingroup\expandafter\endgroup
\expandafter\pxpic@experr\csname pxpic Error:\endcsname

(End definition for \pxpic@experr.)
With the expandable error throwing mechanism out of the way, the following is straightforward again.

\newcommand{\pxpic@experr@noval}[1]{{\pxpic@experr{Missing colour definition for '#1'}}}

(End definition for \pxpic@experr@noval.)
Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

P
\px ......................... 6, 57, 171, 179, 206, 211
\pxpic ........................ 2, 51, 76
\pxpicaddcolorlist .......... 6, 103
\pxpicforget .................. 6, 97
\pxpicHT .................. 7, 4, 13, 14, 59, 61, 72, 73, 76, 195
\pxpiclogo ................... 7, 69
\pxpicnewcolorlist ............ 6, 103
\pxpicnewmode ............... 6, 99, 175
\pxpicsetcolorlist ............. 6, 103
\pxpicsetmode ............... 6, 99
\pxpicsetup .................. 4, 54, 68
\pxpicWD ....... 7, 4, 13, 15, 60, 73, 195, 196
\pxskip ..................... 6, 17, 58, 163, 170, 178
\pxpic@inner@box ....... 9, 10, 38, 43, 48, 55
\pxpic@kern ................ 8, 16, 59, 60, 62, 74, 142, 143, 146, 150, 155
\pxpic@linedone ........... 147
\pxpic@openbrace ........... 138, 217, 218
\pxpic@parse ................. 63, 139
\pxpic@parse@px .... 25, 151, 155, 157
\pxpic@parse@px@cmy .... 173
\pxpic@parse@px@cmyk .... 173
\pxpic@parse@px@Gray .... 173
\pxpic@parse@px@gray .... 173
\pxpic@parse@px@HSB .... 173
\pxpic@parse@px@Hsb .... 173
\pxpic@parse@px@hsb .... 173
\pxpic@parse@px@HTML .... 173
\pxpic@parse@px@named .... 167
\pxpic@parse@px@named .... 157
\pxpic@parse@px@named .... 173
\pxpic@parse@px@RGB .... 173
\pxpic@parse@px@RGB .... 173
\pxpic@parse@px@rgb .... 173
\pxpic@parse@px@tHsb .... 173
\pxpic@parse@px@tHsb .... 173
\pxpic@parse@px@wave .... 173
\pxpic@parse@px@wave .... 173
\pxpic@parse@line .... 143, 147
\pxpic@px ................... 57, 195
\pxpic@setcolor ........... 18, 197
\pxpic@setcolor@colorlist ... 197, 217
\pxpic@setcolor@colorlist ... 197, 218
\pxpic@setcolor@colorlist ... 113, 127, 213
\pxpic@setcolorlist ....... 103
\pxpic@skip ................. 58, 195
\pxpic@tmp ................. 126, 128, 131
\pxpic@undef .............. 133, 173, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194
\pxpic@undef .............. 98, 194