

PixelArt *

A package to draw pixel-art pictures.

Louis Paternault
spalax+ctan(at)gresille(dot)org

December 5, 2017

Abstract

This package defines macros to draw pixel-art pictures using L^AT_EX.

Contents

1	Introduction	2
1.1	License	2
1.2	Overview	2
2	Download and Install	2
2.1	Manual installation	2
3	Usage	2
3.1	Package options	2
3.2	Macros	3
4	Bugs, Ideas, Undefined behaviours	4
4.1	Missing <code>\pixelart</code> macro	4
4.2	It's insanely sloooooow.	5
4.3	Package option <code>draft</code>	5
4.4	Black and white	5
4.5	Spaces	6
4.6	Leading space	7
4.7	Uneven lines	7
5	Implementation	7
	Change History	9

*This document corresponds to `pixelart` v0.1.0, dated 2017/12/05. Home page, bug requests, etc. at <http://framagit.org/spalax/pixelart>

1 Introduction

This document introduces the `pixelart` package, used to draw pixel-art pictures.

1.1 License

This work may be distributed and/or modified under the conditions of the L^AT_EXProject Public License, either version 1.3 of this license or (at your option) any later version.

Further information can be found in the `.dtx` file used to build this document.

1.2 Overview

Installation instructions are given in section 2. Documentation about how to use this package (and examples) is given in section 3. Section 4 lists some known bugs and limitations, and implementation is given in section 5.

2 Download and Install

2.1 Manual installation

- Download the latest archive :

Stable version <http://mirrors.ctan.org/install/macros/latex/contrib/pixelart.tds.zip>

Development version <https://framagit.org/spalax/pixelart/repository/archive.zip?ref=master>

- Unzip the archive.
- If you got the archive from CTAN (stable version), move file `tex/latex/pixelart/pixelart.sty` in a L^AT_EX path.
- If you got the development version, `cd` to the main archive directory, and run `latex pixelart.ins` to build `pixelart.sty`. Move this file into a L^AT_EX path.

3 Usage

3.1 Package options

This package has no package options.

3.2 Macros

This package defines two macros : `\bwpixelart`, used to insert a pixel-art picture, and `\tikzbwpixelart`, which has the same purpose, excepted that it is called from within a `tikzpicture` environment.

3.2.1 `\bwpixelart`

`\bwpixelart` To insert a pixel-art picture in your text, use :

`\bwpixelart[$\langle color, raise, scale \rangle$]{ $\langle pixels \rangle$ }`


Its optional arguments are:

color=black Foreground color (the background is transparent);

scale=1 Scale. By default, a pixel is the size of a `tikzpicture` default unit, which is probably bigger than what you want.

raise=0pt Raise the picture. By default, the bottom of the picture is on the baseline. You might want to lower it a little by giving this option a negative argument.

Its mandatory argument is the picture pixels, as 0's and 1's. Line breaks in this argument are interpreted as line breaks in the pixel art pictures. How spaces are interpreted is undefined (see section 4.5 for more information).

For instance, this heart  was drawn using the following code:

```
1 \bwpixelart[color=red, scale=.05, raise=-1ex]{%
2 001101100
3 011111110
4 111111111
5 111111111
6 111111111
7 011111110
8 001111100
9 000111000
10 000010000
11 }
```

3.2.2 `\tikzbwpixelart`

`\tikzbwpixelart` The second macro, `\tikzbwpixelart` is almost identical to the first one, excepted that it is meant to be called from inside a `tikzpicture` environment. Actually, `\bwpixelart{0101}` is more or less equivalent to calling :


```
1 \begin{tikzpicture}
2   \tikzbwpixelart{0101}
3 \end{tikzpicture}
```

The signature of this macro is :

`\tikzbwpixelart[⟨color, scale⟩]{⟨coordinates⟩}{⟨pixels⟩}`

Its optional arguments are `color` and `scale`, used to set the color and scale of the picture.

Its first mandatory argument is the coordinate of the top left corner of the picture; the second one is the list of pixel (using the same syntax as teh `\bwpixelart` macro).

For instance, this heart  was drawn using the following code:

```

1 \begin{tikzpicture}[scale=.05, baseline=-1em]
2   \fill[red] (5, -5) circle (6.5);
3   \tikzbwpixelart{(0, 0)}{%
4     0011001100
5     0111111110
6     1111111111
7     1111111111
8     1111111111
9     0111111110
10    0011111100
11    0001111000
12    0000110000
13  }
14 \end{tikzpicture}

```

4 Bugs, Ideas, Undefined behaviours

I have great ideas about what this package could do, but:

- I do not need them;
- I am not sure there is a huge *need* for some pixel-art package;
- I have a full-time job, my wife has a far-more-than-full-time job, my daughter *is* a full-time job, so I have very little time to hack...

So, I am listing here some known bugs, undefined behaviours, limitations. You, random passer-by, will be greatly welcome if you were to fix or implement stuff listed in this section. ☺

4.1 Missing `\pixelart` macro

There is no `\pixelart` macro. This is on purpose: given that this package is more or less a working draft, I did not want to register a badly designed `\pixelart` macro. This means that some folk wanting to improve this package can extend the `\bwpixelart` macro and use the name `\pixelart` to fix my design mistakes.

4.2 It's insanely sloooooow.

That's it. It takes almost 30 seconds to compile a document containing only a 128×128 picture (about 16 000 pixels). I have no idea how to fix it. Good luck.

4.3 Package option **draft**

To (partially) fix the previous bug (sloooooow), one could imagine adding a package option **draft**. With this option set, the pixel-art pictures would not be compiled, but would be replaced either by a dummy picture of an arbitrary size, or a dummy picture of the actual size of the picture, if an argument **size**=(4, 5) has been provided to the macro.

This should not be that hard to implement. Patches welcome! ☺

4.4 Black and white

Right now, it is black and white only (or, to be more accurate, any single color on a transparent background).

One *could* produce colored pixel-art pictures, but... it's complicated. For instance, this heart (borrowed from the Django project¹):



could be produced using the following code. Basically (given that colors **violet1** to **violet5** have been correctly defined), we stack up several single-color pixel-art pictures.

```
1 \begin{tikzpicture}[scale=.1]
2   \tikzbpixelart[color=violet1]{(0, 0)}{%
3     0000000
4     0000010
5   }
6   \tikzbpixelart[color=violet2]{(0, 0)}{%
7     0000000
8     0110100
9     0000000
10    0010100
11    0000000
12    0001000
13  }
14  \tikzbpixelart[color=violet3]{(0, 0)}{%
15    0000010
16    0000000
17    1000010
18    0000000
19    0001000
20  }
```

¹<https://www.djangoproject.com/>

```

21 \tikzbpixelart[color=violet4]{(0, 0)}{%
22 0010100
23 1001000
24 0110100
25 0001010
26 0010100
27 }
28 \tikzbpixelart[color=violet5]{(0, 0)}{%
29 0100000
30 0000001
31 0001001
32 0100000
33 }
34 \end{tikzpicture}

```

One could imagine a simpler syntax: we assign several colors to characters, and we use 1, 2, 3, etc. as the pixels to define the picture. This would give the following code.

```

1 \begin{tikzpicture}[scale=.1]
2 \tikzpixelart[colors={
3 1=violet1,
4 2=violet2,
5 3=violet3,
6 4=violet4,
7 5=violet5,
8 }]{(0, 0)}{%
9 0540430
10 4224215
11 3445435
12 0524240
13 0043400
14 0002000
15 }
16 \end{tikzpicture}

```

Once again, patches are welcome! 😊

4.5 Spaces

Spaces are interpreted as line breaks. For instance, this heart ♥ could be written as :

```

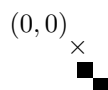
1 \bwpixelart[scale=.03, raise=-1pt]{%
2 0011001100 0111111110 1111111111
3 1111111111 1111111111 0111111110
4 0011111100 0001111000 0000110000
5 }

```

This will work (right now), but is an undefined behaviour, and might change in a later version without prior notice.

4.6 Leading space

As described in the previous section, spaces are considered as line breaks. A problem is that the *first* space(s) is considered as a line breaks as well. Thus, the first space should be commented out (using %), to avoid the case described in the following picture, where the cross marks the location of the upper left corner of the picture, as intended (but not as produced).



```
1 \begin{tikzpicture}[scale=.2]
2   \tikzbpixelart{(0, 0)}{
3     10
4     01
5   }
6   \draw (0, 0) node{\texttimes} node[above left]{$(0, 0)$};
7 \end{tikzpicture}
```

To circumvent this bug, line 2 should have been written as (note the trailing percent character):

```
2   \tikzbpixelart{(0, 0)}{%
```

4.7 Uneven lines

Right now, lines do not *have* to have the same number of characters. For instance, the following heart ♥ could be written as :

```
1 \bwpixelart[scale=.03, raise=-1pt]{%
2   00110011
3   011111111
4   1111111111
5   1111111111
6   1111111111
7   011111111
8   00111111
9   0001111
10  000011
11 }
```

This is an undefined behaviour and might raise an error in the future.

5 Implementation

Load some packages.

```
1 \RequirePackage{pgf}
2 \usepgfmodule{parser}
```

```

3 \RequirePackage{tikz}
4 \usetikzlibrary{calc}
5 \RequirePackage{pgfkeys}
6 \RequirePackage{etoolbox}
7
8   Let the nasty stuff begin.
9
10  Define macro arguments
11
12  \pgfkeys{
13    /PIXELART/BWPIXELART/.is family,
14    /PIXELART/BWPIXELART,
15    scale/.value required,
16    scale/.code={\pgfkeyssetvalue{/PIXELART/BWPIXELART/scale}{#1}},
17    scale=1,
18    raise/.code={\pgfkeyssetvalue{/PIXELART/BWPIXELART/raise}{#1}},
19    raise/.value required,
20    raise=0pt,
21    color/.value required,
22    color/.code={\pgfkeyssetvalue{/PIXELART/BWPIXELART/color}{#1}},
23    color=black,
24  }
25
26  \pgfkeys{
27    /PIXELART/TIKZBWPIXELART/.is family,
28    /PIXELART/TIKZBWPIXELART,
29    scale/.value required,
30    scale/.code={\pgfkeyssetvalue{/PIXELART/TIKZBWPIXELART/scale}{#1}},
31    scale=1,
32    color/.value required,
33    color/.code={\pgfkeyssetvalue{/PIXELART/TIKZBWPIXELART/color}{#1}},
34    color=black,
35  }
36
37  Define argument parser, parsing the sequence of 0 and 1 defining the pixel art.
38
39  \pgfparserdef{@bwpixelart}{initial}{blank space \space}{
40    \@bwpa@newline
41  }
42  \pgfparserdef{@bwpixelart}{initial}{the character 0}{
43    \coordinate (@pixelart) at ($(@pixelart) + (1, 0)$);
44  }
45  \pgfparserdef{@bwpixelart}{initial}{the character 1}{
46    \fill (@pixelart) rectangle ++(1, 1);
47    \coordinate (@pixelart) at ($(@pixelart) + (1, 0)$);
48  }
49  \pgfparserdef{@bwpixelart}{initial}{the letter @}{
50    \pgfparserswitch{final}
51  }

```



```

47 }
48
49 \newcommand{\@bwpa@newline}{
50   \coordinate (@pixelart) at ($(@pixelart@startline) + (0, -1)$);
51   \coordinate (@pixelart@startline) at (@pixelart);
52 }
53
\bwpxelart Define \bwpxelart, used to draw black-and-white pixelart.
54 \newcommand{\bwpxelart}[2][]{\%
55   \pgfkeys{/PIXELART/BWPIXELART, #1}%
56   \raisebox{\pgfkeysvalueof{/PIXELART/BWPIXELART/raise}}{\%
57     \tikz[
58       scale=\pgfkeysvalueof{/PIXELART/BWPIXELART/scale},
59     ]{
60       \tikzbwpxelart[%
61         color=\pgfkeysvalueof{/PIXELART/BWPIXELART/color},
62       ]{(0, 0)}{#2}
63     }%
64   }%
65 }}
66
\tikzbwpxelart Define \tikzbwpxelart, used to draw black-and-white pixelart from inside a
tikzpicture environment.
67 \newcommand{\tikzbwpxelart}[3][]{
68   \pgfkeys{/PIXELART/TIKZBWPIXELART, #1}
69   \begin{scope}[shift={#2}]
70     \begin{scope}[#1]
71       \coordinate (@pixelart) at (0, 0);
72       \coordinate (@pixelart@startline) at (@pixelart);
73
74       \pgfparserparse{@bwpxelart} #3 @
75     \end{scope}
76   \end{scope}
77 }
78
That's all, folks!
79 \makeatother
80

```

Index

Numbers written in *italic* refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in *roman* refer to the code lines where the entry is used.

B

\bwpxelart 54

T
`\tikzbpixelart . 60, 67`