pst-fill
A PSTricks package for filling and tiling areas

Timothy Van Zandt*

March 11, 2007— Version 1.00
Documentation revised March 11, 2007

Contents

*tvz@econ.insead.fr. (documentation by Denis Girou (Denis.Girou@idris.fr) and Herbert Voß(hvoss@tug.org).

Abstract

'pst-fill' is a PSTricks (13), (6), (14), (9), (7) package to draw easily various kinds of filling and tiling of areas. It is also a good example of the great power and flexibility of PSTricks, as in fact it is a very short program (it body is around 200 lines long) but nevertheless really powerful.

It was written in 1994 by Timothy VAN ZANDT but publicly available only in PSTricks 97 and without any documentation. We describe here the version *97 patch 2* of December 12, 1997, which is the original one modified by myself to manage *tilings* in the so-called *automatic* mode. This article would like to serve both of reference manual and of user's guide.

This package is available on CTAN in the `graphics/pstricks` directory (files `latex/pst-fill.sty` and `generic/pst-fill.tex`).

1 Introduction

Here we will refer as *filling* as the operation which consist to fill a defined area by a pattern (or a composition of patterns). We will refer as *tiling* as the operation which consist to do the same thing, but with the control of the starting point, which is here the upper left corner. The pattern is positioned relatively to this point. This make an essential difference between the two modes, as without control of the starting point we can't draw *tilings* (sometimes called *tesselations*) as used in many fields of Art and Science [1].

Nevertheless, as tilings are a wide and difficult field in mathematics, this package is limited to simple ones, mainly *monohedral* tilings with one prototile (which can be composite, see section 3.1). With some experience and wiliness we can do more and obtained easily rather sophisticated results, but obviously hyperbolic tilings like the famous ESCHER ones or aperiodic tilings like the PENROSE ones are not in the capabilities of this package. For more complex needs, we must used low level and more painfull technics, with the basic `\multido` and `\multirput` macros.

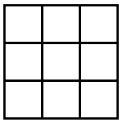
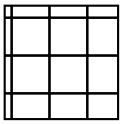
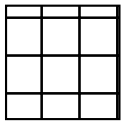
2 Package history and description of it two different modes

As already said, this package was written in 1994 by Timothy VAN ZANDT. Two modes were defined, called respectively *manual* and *automatic*. For both, the pattern is generated on contiguous positions in a rather large area which include the region to fill, later cut to the required dimensions by clipping mechanism. In

[1] For an extensive presentation of tilings, in their history and usage in many fields, see the reference book (8).

In the TEX world, few work was done on tilings. You can look at the *tile* extension of the XY-pic package (10), at the articles of Kees VAN DER LAAN (11, paragraph 7) (the tiling was in fact directly done in PostScript) and (12), at the METAPOST program (available on `graphics/metapost/contrib/macros/truchet`) by Denis ROEGEL for the TRUCHET contest in 1995 (5) and at the METAPOST package (2) to draw patterns, which have a strong connection with tilings.

the first mode, the pattern is explicitely inserted in the PostScript file each time. In the second one, the result is the same but with an unique explicit insertion of the pattern and a repetition done by PostScript. Nevertheless, in this method, the control of the starting point was loosed, so it allowed only to *fill* a region and not to *tile* it.

See the difference between the two modes, *tiling*:  and *filling*:  or  as we can see that initial position is arbitrary and dependent of the current point.

It's clear that usage of filling is very restrictive comparing to tiling, as desired effects required very often the possibility to control the starting point. So, this mode was of limited interest, but unfortunately the *manual* one has the very big disadvantage to require very huge amounts of ressources, mainly in disk space and consequently in printing time. A small tiling can require sometimes several megabytes in *manual* mode! So, it was very often not really usable in practice.

It is why I modified the code, to allow tilings in *automatic* mode, controlling in this mode too the starting point. And most of the time, that is to say if some special options are not used, the tiling is done exactly in the region described, which make it faster. So there is no more reason to use the *manual* mode, apart very special cases where *automatic* one cannot work, as explained later – currently, I know only one case.

To load this modified *automatic* mode, with LaTeX use simply:
`\usepackage[tiling]{pst-fill}`
and in plain TeX after:
`\input{pst-fill}`
add the following definition:
`\def\PstTiling{true}`

To obtain the original behaviour, just don't use the *tiling* optional keyword at loading.

Take care than in *tiling* mode, I introduce also some other changes. First I define aliases on some parameter names for consistancy (all specific parameters will begin by the `fill` prefix in this case) and I change some default values, which were not well adapted for tilings (`fillsep` is set to 0 and as explained `fillsize` set to `auto`). I rename `fillcycle` to `fillcyclex`. I also restore normal way so that the frame of the area is drawn and all line (`linestyle`, `linecolor`, `doubleline`, etc.) parameters are now active (but there are not in non *tiling* mode). And I also introduce new parameters to control the tilings (see below).

In all the following examples, we will consider only the *tiling* mode.

To do a tiling, we have just to define the pattern with the `\psboxfill` macro and to use the new `fillstyle boxfill`.

Note that tilings are drawn from left to right and top to bottom, which can have an importance in some circonstances.

PostScript programmers can be also interested to know that, even in the *automatic* mode, the iterations of the pattern are managed directly by the PostScript code of the package which used only PostScript Level 1 operators. The special ones introduced in Level 2 for drawing of patterns (1, section 4.9) are not used.

And first, for conveniance, we define a simple `\Tiling` macro, which will simplify our examples:

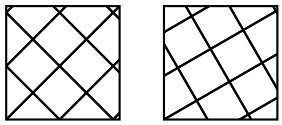
```
\newcommand{\Tiling}[2][]{%
  \edef\Temp{#1}%
  \begin{pspicture}#2
    \ifx\Temp\empty
      \psframe[fillstyle=boxfill]#2
    \else
      \psframe[fillstyle=boxfill,#1]#2
    \fi
  \end{pspicture}}
```

2.1 Parameters

There are **14** specific parameters available to change the way the filling/tiling is defined, and one debugging option.

fillangle (real): the value of the rotation applied to the patterns (*Default: 0*).

In this case, we must force the tiling area to be notably larger than the area to cover, to be sure that the defined area will be covered after rotation.



```
\newcommand{\Square}{%
  \begin{pspicture}(1,1)
    \psframe[dimen=middle](1,1)
  \end{pspicture}}
\psset{unit=0.5}
\psboxfill{\Square}
\Tiling[fillangle=45]{(3,3)}\quad
\Tiling[fillangle=-60]{(3,3)}
```

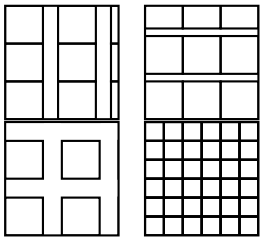
`fillsepx` (real‖dim) : value of the horizontal separation between consecutive patterns (*Default: 0 for tilings*[2]*, 2pt otherwise*).

`fillsepy` (real‖dim): value of the vertical separation between consecutive patterns (*Default: 0 for tilings*[2]*, 2pt otherwise*).

`fillsep` (real‖dim): value of horizontal and vertical separations between consecutive patterns (*Default: 0 for tilings*[2]*, 2pt otherwise*).

These values can be negative, which allow the tiles to overlap.

[2]This option was added by me, is not part of the original package and is available only if the `tiling` keyword is used when loading the package.



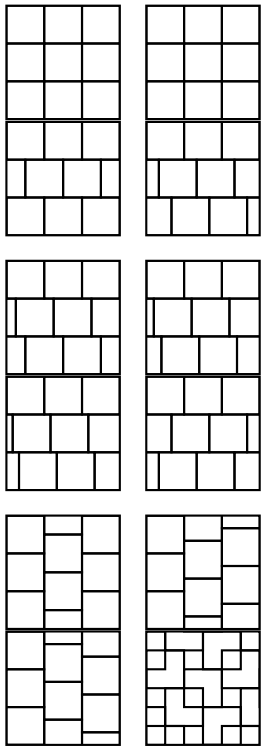
```
\psset{unit=0.5}
\psboxfill{\Square}
\Tiling[fillsepx=2mm]{(3,3)}
\Tiling[fillsepy=1mm]{(3,3)}\\
\Tiling[fillsep=0.5]{(3,3)}
\Tiling[fillsep=-0.5]{(3,3)}
```

`fillcyclex`[3] (integer): Shift coefficient applied to each row (*Default: 0*).

`fillcycley`[2] (integer): Same thing for columns (*Default: 0*).

`fillcycle`[2] (integer): Allow to fix both `fillcyclex` and `fillcycley` directly to the same value (*Default: 0*).

For instance, if `fillcyclex` is 2, the second row of patterns will be horizontally shifted by a factor of $\frac{1}{2} = 0.5$, and by a factor of 0.333 if `fillcyclex` is 3, etc.). These values can be negative.



```
\psset{unit=0.5}
\psboxfill{\Square}
\newcommand{\TilingA}[1]{\Tiling[fillcyclex
  =#1]{(3,3)}}
\TilingA{0} \TilingA{1}\\
\TilingA{2} \TilingA{3}\\[3mm]
\TilingA{4} \TilingA{5}\\
\TilingA{6} \TilingA{-3}\\[3mm]
\Tiling[fillcycley=2]{(3,3)}
\Tiling[fillcycley=3]{(3,3)}\\
\Tiling[fillcycley=-3]{(3,3)}
\Tiling[fillcycle=2]{(3,3)}
```

`fillmovex`[2] (real‖dim): value of the horizontal moves between consecutive patterns (*Default: 0*).

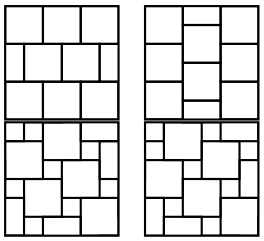
[3] It was `fillcycle` in the original version.

`fillmovey`[2] (real‖dim): value of the vertical moves between consecutive patterns (*Default: 0*).

`fillmove`[2] (real‖dim): value of horizontal and vertical moves between consecutive patterns (*Default: 0*).

These parameters allow the patterns to overlap and to draw some special kinds of tilings. They are implemented only for the *automatic* and *tiling* modes and their values can be negative.

In some cases, the effect of these parameters will be the same that with the `fillcycle?` ones, but you can see that it is not true for some other values.



```
\psset{unit=0.5}
\psboxfill{\Square}
\Tiling[fillmovex=0.5]{(3,3)}
\Tiling[fillmovey=0.5]{(3,3)}\\
\Tiling[fillmove=0.5]{(3,3)}
\Tiling[fillmove=-0.5]{(3,3)}
```

`fillsize` (auto‖{(real‖dim,real‖dim)(real‖dim,real‖dim)}) : The choice of *automatic* mode or the size of the area in *manual* mode. If first pair values are not given, (0,0) is used. (*Default: auto when* tiling *mode is used, (-15cm,-15cm)(15cm,15cm) otherwise*).

As explained in the introduction, the *manual* mode can require very huge amount of computer ressources. So, it usage is to discourage in front off the *automatic* mode. It seems only useful in special circonstances, in fact when the *automatic* mode failed, which is known only in one case, for some kinds of EPS files, as the ones produce by dump of portions of screens (see 3.2).

`fillloopaddx`[2] (integer): number of times the pattern is added on left and right positions (*Default: 0*).

`fillloopaddy`[2] (integer): number of times the pattern is added on top and bottom positions (*Default: 0*).

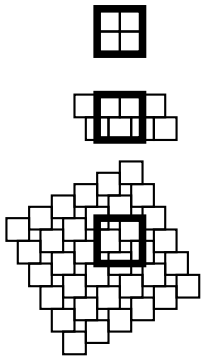
`fillloopadd`[2] (integer): number of times the pattern is added on left, right, top and bottom positions (*Default: 0*).

These parameters are only useful in special circonstances, as for complex patterns when the size of the rectangular box used to tile the area doesn't correspond to the pattern itself (see an example in Figure 3.1) and also sometimes when the size of the pattern is not a divisor of the size of the area to fill and that the number of loop repeats is not properly computed, which can occur.

They are implemented only for the *tiling* mode.

`PstDebug`[2] (integer, 0 or 1): to require to see the exact tiling done, without clipping (*Default: 0*).

It's mainly useful for debugging or to understand better how the tilings are done. It is implemented only for the *tiling* mode.



```
\psset{unit=0.3,PstDebug=1}
\psboxfill{\Square}
\psset{linewidth=1mm}
\Tiling{(2,2)}\\[5mm]
\Tiling[fillcyclex=2]{(2,2)}\\[1cm]
\Tiling[fillmove=0.5]{(2,2)}
```

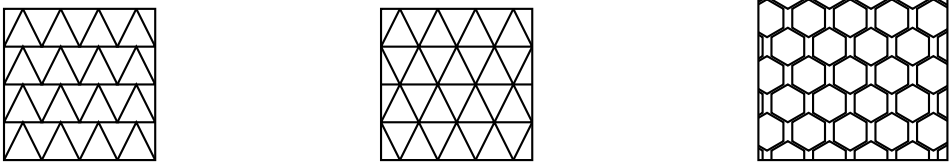
3 Examples

In fact this unique `\psboxfill` macro allow a lot a variations and different usages. We will try here to demonstrate this.

3.1 Kind of tiles

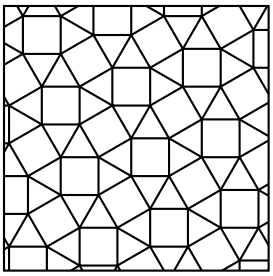
Of course, we can access to all the power of PSTricks macros to define the *tiles* (*patterns*) used. So, we can define complicated ones.

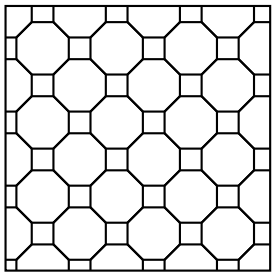
Here we give four other Archimedian tilings (those built with only some regular polygons) among the twelve existing, first discovered completely by Johanes Kepler at the beginning of 17th century (8), the two other *regular* ones with the tiling by squares, formed by a unique regular polygon, and two other formed by two different regular polygons.



```
  \newcommand{\Triangle}{%
    \begin{pspicture}(1,1)
      \pstriangle[dimen=middle](0.5,0)(1,1)
    \end{pspicture}}
  \newcommand{\Hexagon}{
^^A sin(60)=0.866
    \begin{pspicture}(0.866,0.75)
      \SpecialCoor
^^A  Hexagon
      \pspolygon[dimen=middle]%
        (0.5;30)(0.5;90)(0.5;150)(0.5;210)(0.5;270)(0.5;330)
    \end{pspicture}}

  \psset{unit=0.5}
  \psboxfill{\Triangle}
  \Tiling{(4,4)}\hfill
^^A The two other regular tilings
  \Tiling[fillcyclex=2]{(4,4)}\hfill
  \psboxfill{\Hexagon}
  \Tiling[fillcyclex=2,fillloopaddy=1]{(5,5)}
```



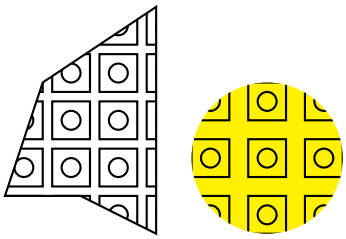


```
\newcommand{\ArchimedianA}{%
   ^^A Archimedian tiling 3^2.4.3.4
  \psset{dimen=middle}
   ^^A sin(60)=0.866
  \begin{pspicture}(1.866,1.866)
    \psframe(1,1)
    \psline(1,0)(1.866,0.5)(1,1)(0.5,1.866)(0,1)(-0.866,0.5)
    \psline(0,0)(0.5,-0.866)
  \end{pspicture}}
\newcommand{\ArchimedianB}{%
   ^^A Archimedian tiling 4.8^2
  \psset{dimen=middle,unit=1.5}
   ^^A sin(22.5)=0.3827 ; cos(22.5)=0.9239
  \begin{pspicture}(1.3066,0.6533)
    \SpecialCoor
   ^^A Octogon
    \pspolygon(0.5;22.5)(0.5;67.5)(0.5;112.5)(0.5;157.5)
              (0.5;202.5)(0.5;247.5)(0.5;292.5)(0.5;337.5)
  \end{pspicture}}

\psset{unit=0.5}
\psboxfill{\ArchimedianA}
\Tiling[fillmove=0.5]{(7,7)}\hfill
\psboxfill{\ArchimedianB}
\Tiling[fillcyclex=2,fillloopaddy=1]{(7,7)}
```

We can of course tile an area arbitrarily defined. And with the `addfillstyle` parameter[4], we can easily mix the `boxfill` style with another one.

[4]Introduced in PSTricks 97.

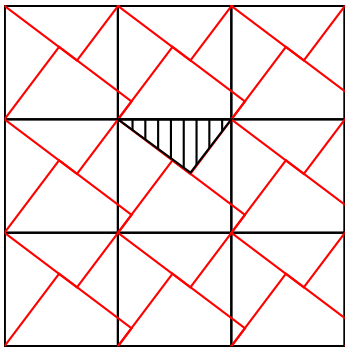


```
\psset{unit=0.5,dimen=middle}
\psboxfill{%
  \begin{pspicture}(1,1)
    \psframe(1,1)
    \pscircle(0.5,0.5){0.25}
  \end{pspicture}}
\begin{pspicture}(4,6)
  \pspolygon[fillstyle=boxfill,
    fillsep=0.25](0,1)(1,4)(4,6)(4,0)
    (2,1)
\end{pspicture}\hspace{1em}
\begin{pspicture}(4,4)
   \pscircle[linestyle=none,fillstyle
     =solid,fillcolor=yellow,fillsep
     =0.5,
            addfillstyle=boxfill
              ](2,2){2}
\end{pspicture}
```

Various effects can be obtained, sometimes complicated ones very easily, as in this example reproduced from one shown by Slavik JABLAN in the field of *OpTiles*, inspired by the *Op-art*:

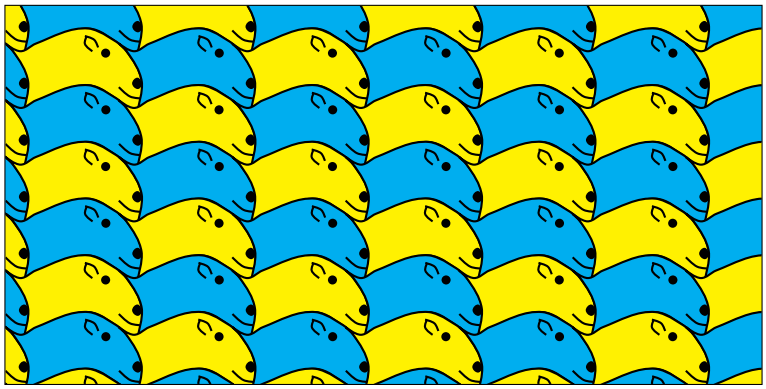
```
\newcommand{\ProtoTile}{%
 \begin{pspicture}(1,1)%%% 1/12=0.08333
  \psset{linestyle=none,linewidth=0,
    hatchwidth=0.08333\psunit,hatchsep=0.08333\psunit}
  \psframe[fillstyle=solid,fillcolor=black,addfillstyle=hlines,hatchcolor=white
    ](1,1)
  \pswedge[fillstyle=solid,fillcolor=white,addfillstyle=hlines]{1}{0}{90}
 \end{pspicture}}
\newcommand{\BasicTile}{%
 \begin{pspicture}(2,1)
   \rput[lb](0,0){\ProtoTile}\rput[lb](1,0){\psrotateleft{\ProtoTile}}
 \end{pspicture}}
\ProtoTile\hfill\BasicTile\hfill
\psboxfill{\BasicTile}
\Tiling[fillcyclex=2]{(4,4)}
```

It is also directly possible to surimpose several different tilings. Here is the splendid visual proof of the PYTHAGORE theorem done by the arab mathematician ANNAIRIZI around the year 900, given by superposition of two tilings by squares of different sizes.



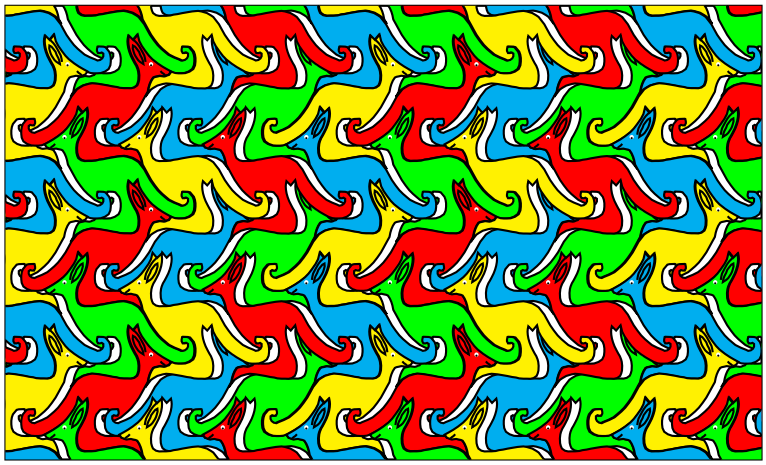
```
\psset{unit=1.5,dimen=middle}
\begin{pspicture*}(3,3)
  \psboxfill{\begin{pspicture}(1,1)
    \psframe(1,1)\end{pspicture}}
  \psframe[fillstyle=boxfill](3,3)
  \psboxfill{\begin{pspicture}(1,1)
    \rput{-37}{\psframe[linecolor=red](0.8,0.8)}
  \end{pspicture}}
  \psframe[fillstyle=boxfill](3,4)
  \pspolygon[fillstyle=hlines,hatchangle=90](1,2)(1.64,1.53)(2,2)
\end{pspicture*}
```

In a same way, it is possible to build tilings based on figurative patterns, in the style of the famous Escher ones. Following an example of André Deledicq (4), we first show a simple tiling of the *p1* category (according to the international classification of the 17 symmetry groups of the plane first discovered by the russian crystalographer Jevgraf Fedorov at the end of the 19th century):



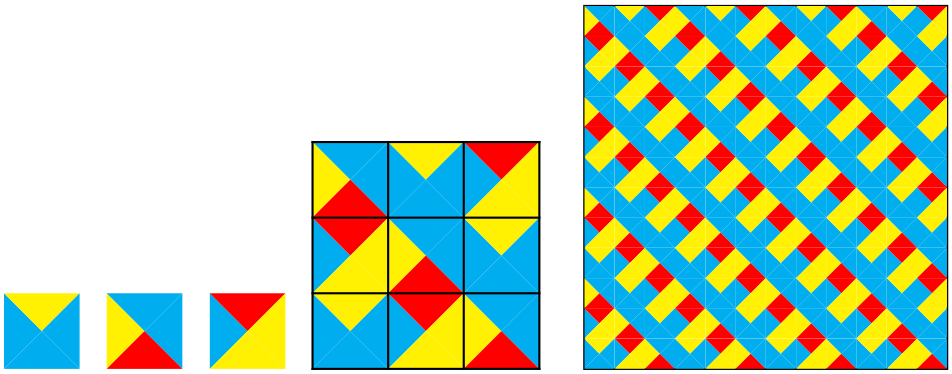
```
\newcommand{\SheepHead}[1]{%
  \begin{pspicture}(3,1.5)
    \pscustom[liftpen=2,fillstyle=solid,fillcolor=#1]{%
      \pscurve(0.5,-0.2)(0.6,0.5)(0.2,1.3)(0,1.5)(0,1.5)
        (0.4,1.3)(0.8,1.5)(2.2,1.9)(3,1.5)(3,1.5)(3.2,1.3)
        (3.6,0.5)(3.4,-0.3)(3,0)(2.2,0.4)(0.5,-0.2)}
    \pscircle*(2.65,1.25){0.12\psunit} % Eye
    \psccurve*(3.5,0.3)(3.35,0.45)(3.5,0.6)(3.6,0.4)% Muzzle
   ^^A   % Mouth
      \pscurve(3,0.35)(3.3,0.1)(3.6,0.05)
   ^^A   % Ear
      \pscurve(2.3,1.3)(2.1,1.5)(2.15,1.7)\pscurve(2.1,1.7)(2.35,1.6)(2.45,1.4)
  \end{pspicture}}
\psboxfill{\psset{unit=0.5}\SheepHead{yellow}\SheepHead{cyan}}
\Tiling[fillcyclex=2,fillloopadd=1]{(10,5)}
```

Now a tiling of the *pg* category (the code for the kangaroo itself is too long to be shown here, but has no difficulties ; the kangaroo is reproduce from an original picture from Raoul Raba and here is a translation in PSTricks from the one drawn by Emmanuel Chailloux and Guy Cousineau for their MLgraph system (3)):



```
\psboxfill{\psset{unit=0.4}
  \Kangaroo{yellow}\Kangaroo{red}\Kangaroo{cyan}\Kangaroo{green}%
  \psscalebox{-1 1}{%
    \rput(1.235,4.8){\Kangaroo{green}\Kangaroo{cyan}\Kangaroo{red}\Kangaroo{
      yellow}}}}
  \Tiling[fillloopadd=1]{(10,6)}
```

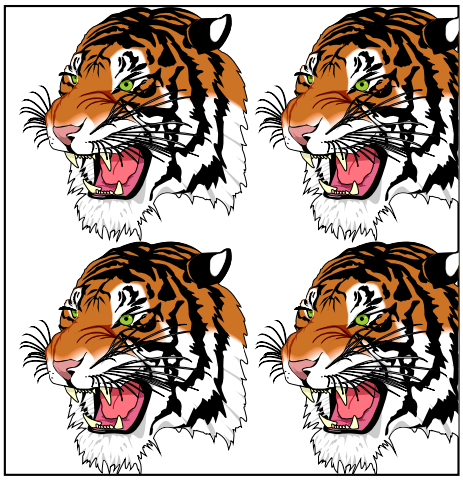
And here a WANG tiling (16), (8, chapter 11), based on very simple tiles of the form of a square and composed of four colored triangles. Such tilings are built with only a matching color constraint. Despite of it simplicity, it is an important kind of tilings, as WANG and others used them to study the special class of *aperiodic* tilings, and also because it was shown that surprisingly this tiling is similar to a TURING machine.



```
\newcommand{\WangTile}[4]{%
  \begin{pspicture}(1,1)
    \pspolygon*[linecolor=#1](0,0)(0,1)(0.5,0.5)
    \pspolygon*[linecolor=#2](0,1)(1,1)(0.5,0.5)
    \pspolygon*[linecolor=#3](1,1)(1,0)(0.5,0.5)
    \pspolygon*[linecolor=#4](1,0)(0,0)(0.5,0.5)
  \end{pspicture}}
\newcommand{\WangTileA}{\WangTile{cyan}{yellow}{cyan}{cyan}}
\newcommand{\WangTileB}{\WangTile{yellow}{cyan}{cyan}{red}}
\newcommand{\WangTileC}{\WangTile{cyan}{red}{yellow}{yellow}}
\newcommand{\WangTiles}[1][]{%
  \begin{pspicture}(3,3) \psset{ref=lb}
    \rput(0,2){\WangTileB} \rput(1,2){\WangTileA}%
    \rput(2,2){\WangTileC} \rput(0,1){\WangTileC}%
    \rput(1,1){\WangTileB} \rput(2,1){\WangTileA}
    \rput(0,0){\WangTileA} \rput(1,0){\WangTileC}%
    \rput(2,0){\WangTileB}
    #1
  \end{pspicture}}
\WangTileA\hfill\WangTileB\hfill\WangTileC\hfill
\WangTiles[{\psgrid[subgriddiv=0,gridlabels=0](3,3)}]\hfill
\psset{unit=0.4} \psboxfill{\WangTiles} \Tiling{(12,12)}
```

3.2 External graphic files

We can also fill an arbitrary area with an external image. We have only, as usual, to matter of the *BoundingBox* definition if there is no one provided or if it is not the accurate one, as for the well known `tiger` picture part of the `ghostscript` distribution.



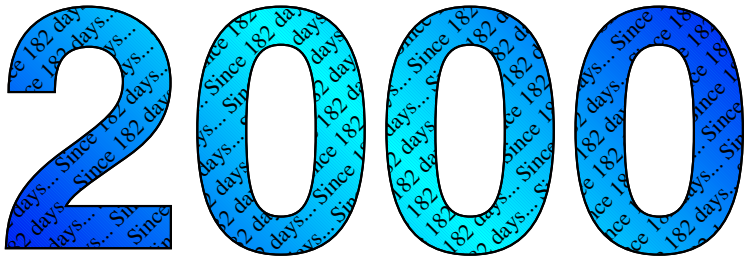
```
\psboxfill{%% Strangely require x1=x2...
  \begin{pspicture}(0,1)(0,4.1)
    \includegraphics[bb=17 176 560 74,width=3cm]{tiger}
  \end{pspicture}}
\Tiling{(6,6.2)}
```

Nevertheless, there are some special files for which the *automatic* mode doesn't work, specially for some files obtained by a screen dump, as in the next example, where a picture was reduced before it conversion in the *Encapsulated PostScript* format by a screen dump utility. In this case, usage of the *manual* mode is the only alternative, at the price of the real multiple inclusion of the EPS file. We must take care to specify the correct `fillsize` parameter, because otherwise the default values are large and will load the file many times, perhaps just really using few occurrences as the other ones would be clipped...

```
\psboxfill{\includegraphics{flowers}}
\begin{pspicture}(8,4)
  \psellipse[fillstyle=boxfill,fillsize={(8,4)}](4,2)(4,2)
\end{pspicture}
```

3.3 Tiling of characters

We can also use the `\psboxfill` macro to fill the interior of characters for special effects like these ones:



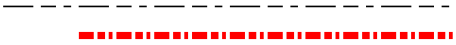
```
\DeclareFixedFont{\bigsf}{T1}{phv}{b}{n}{4.5cm}
\DeclareFixedFont{\smallrm}{T1}{ptm}{m}{n}{3mm}
\psboxfill{\smallrm Since 182 days...}
\begin{pspicture*}(8,4)
  \centerline{%
    \pscharpath[fillstyle=gradient,gradangle=-45,
                gradmidpoint=0.5,addfillstyle=boxfill,
                fillangle=45,fillsep=0.7mm]
               {\rput[b](0,0.1){\bigsf 2000}}}
\end{pspicture*}
```

```
\DeclareFixedFont{\mediumrm}{T1}{ptm}{m}{n}{2cm}
\psboxfill{%
  \psset{unit=0.1,linewidth=0.2pt}
  \Kangaroo{PeachPuff}\Kangaroo{PaleGreen}%
    \Kangaroo{LightBlue}\Kangaroo{LemonChiffon}%
  \psscalebox{-1 1}{%
    \rput(1.235,4.8){%
      \Kangaroo{LemonChiffon}\Kangaroo{LightBlue}%
        \Kangaroo{PaleGreen}\Kangaroo{PeachPuff}}}}
^^A  % A kangaroo of kangaroos...
\begin{pspicture}(8,2)
  \pscharpath[linestyle=none,fillstyle=boxfill,fillloopadd=1]
             {\rput[b](4,0){\mediumrm Kangaroo}}
\end{pspicture}
```

3.4 Other kinds of usage

Other kinds of usage can be imagined. For instance, we can use tilings in a sort of degenerated way to draw some special lines made by a unique or multiple repeating

patterns. But it can be only a special dashed line, as here with three different dashes:



```
\newcommand{\Dashes}{%
  \psset{dimen=middle}
  \begin{pspicture}(0,-0.5\pslinewidth)(1,0.5\pslinewidth)
    \rput(0,0){\psline(0.4,0)}%
      \rput(0.5,0){\psline(0.2,0)}%
      \rput(0.8,0){\psline(0.1,0)}
  \end{pspicture}}

\newcommand{\SpecialDashedLine}[3]{%
  \psboxfill{#3}
  \Tiling[linestyle=none]
        {(#1,-0.5\pslinewidth)(#2,0.5\pslinewidth)}}

\SpecialDashedLine{0}{7}{\Dashes}

\psset{unit=0.5,linewidth=1mm,linecolor=red}
\SpecialDashedLine{0}{10}{\Dashes}
```

It allow also to use special patterns in business graphics, as in the following example generated by `PstChart`[5].

4 "Dynamic" tilings

In some cases, tilings used non *static* tiles, that is to say that the *prototile(s)*, even if unique, can have several forms, by instance specified by different colors or rotations, not fixed before generation or varying each time.

4.1 Lewthwaite-Pickover-Truchet tiling

We give here for example the so-called *Truchet* tiling, which much be in fact better called *Lewthwaite-Pickover-Truchet (LPT)* tiling [6].

The unique prototile is only a square with two opposite circle arcs. This tile has obviously two positions, if we rotate it from 90 degrees (see the two tiles on the next figure). A *LPT tiling* is a tiling with randomly oriented LPT tiles. We can see that even if it is very simple in it principle, it draw sophisticated curves with strange properties.

Nevertheless, in the straightforward way 'pst-fill' does not work, because the `\psboxfill` macro store the content of the tile used in a TEX box, which is static. So the calling to the random function is done only one time, which explain that only one rotation of the tile is used for all the tiling. It's only the one of the two rotations which could differ from one drawing to the next one...

[5] A personal development to draw business charts with PSTricks, not distributed.

[6] For description of the context, history and references about Sébastien TRUCHET and this tiling, see (5).

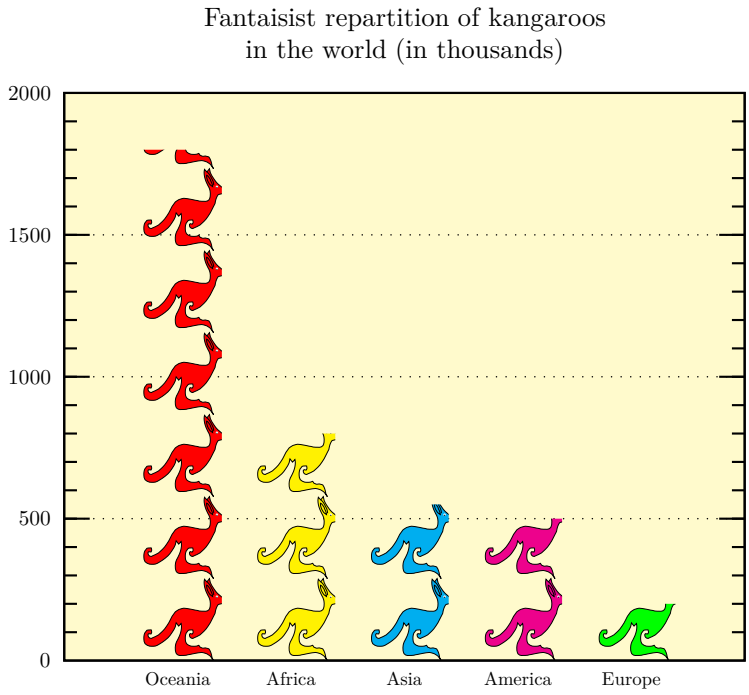
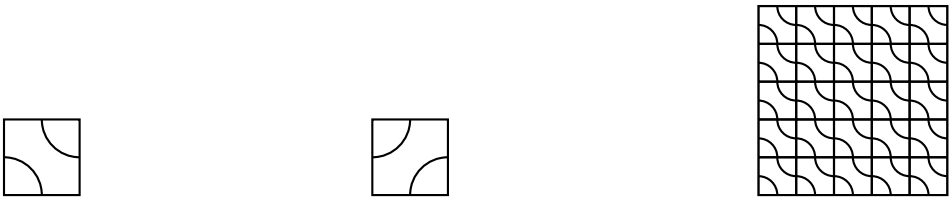


Figure 1: Bar chart generated by PstChart, with bars filled by patterns



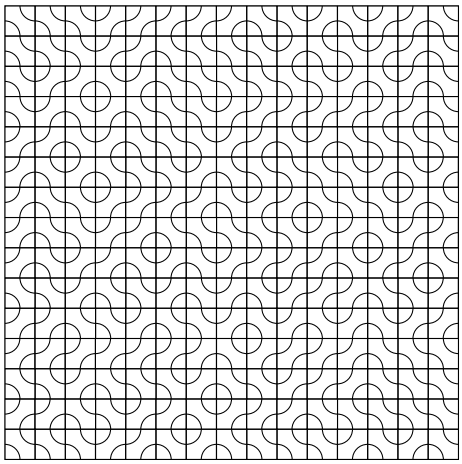
```
^^A   % LPT prototile
  \newcommand{\ProtoTileLPT}{%
    \psset{dimen=middle}
    \begin{pspicture}(1,1)
      \psframe(1,1)
      \psarc(0,0){0.5}{0}{90}
      \psarc(1,1){0.5}{-180}{-90}
    \end{pspicture}}

^^A   % LPT tile
  \newcount\Boolean
  \newcommand{\BasicTileLPT}{%
^^A     % From random.tex by Donald Arseneau
    \setrannum{\Boolean}{0}{1}%
    \ifnum\Boolean=0
      \ProtoTileLPT%
    \else
      \psrotateleft{\ProtoTileLPT}%
    \fi}

  \ProtoTileLPT\hfill\psrotateleft{\ProtoTileLPT}\hfill
  \psset{unit=0.5}
  \psboxfill{\BasicTileLPT}
  \Tiling{(5,5)}
```

But, for simple cases, there is a solution to this problem using a mixture of PSTricks and PostScript programming. Here the PSTricks construction `\pscustom{\code{...}}` allow to insert PostScript code inside the LaTeX + PSTricks one.

Programmation is less straightforward, but it has also the advantage to be notably faster, as all the tilings operations are done in PostScript, and mainly to not be limited by TeX memory (the TeX + PSTricks solution I wrote in 1995 for the colored problem was limited to small sizes for this reason). Just note also that `\pslbrace` and `\psrbrace` are two PSTricks macros to define and be able to insert the `{` and `}` characters.



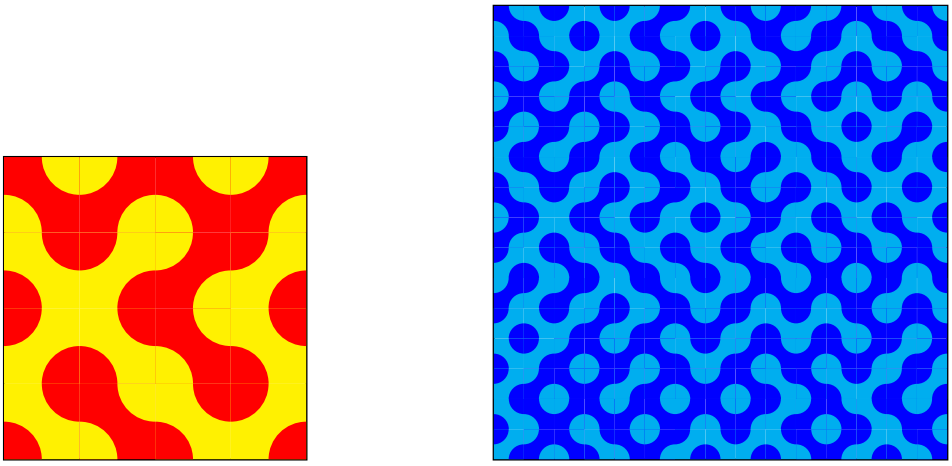
```
^^A  % LPT prototile
  \newcommand{\ProtoTileLPT}{%
    \psset{dimen=middle}
    \psframe(1,1)
    \psarc(0,0){0.5}{0}{90}
    \psarc(1,1){0.5}{-180}{-90}}

^^A  % Counter to change the random seed
  \newcount\InitCounter
^^A  % LPT tile
  \newcommand{\BasicTileLPT}{%
    \InitCounter=\the\time
    \pscustom{\code{%
      rand \the\InitCounter\space sub 2 mod 0 eq \pslbrace}}
    \begin{pspicture}(1,1)
      \ProtoTileLPT
    \end{pspicture}%
    \pscustom{\code{\psrbrace \pslbrace}}
    \psrotateleft{\ProtoTileLPT}%
    \pscustom{\code{\psrbrace ifelse}}}

  \psset{unit=0.4,linewidth=0.4pt}
  \psboxfill{\BasicTileLPT}
  \Tiling{(15,15)}
```

Using the very surprising fact (see (5)) that coloration of these tiles do not depend of their neighbors (even if it is difficult to believe as the opposite seems obvious!) but only of the parity of the value of row and column positions, we can directly program in the same way a colored version of the LPT tiling.

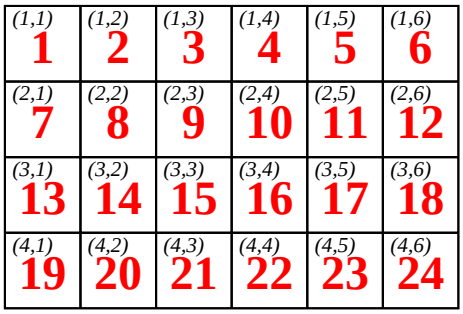
We have also introduce in the 'pst-fill' code for *tiling* mode two new accessible PostScript variables, `row` and `column`[2], which can be useful in some circonstances, like this one.



```
^^A  % LPT prototile
 \newcommand{\ProtoTileLPT}[2]{%
   \psset{dimen=middle,linestyle=none,fillstyle=solid}
   \psframe[fillcolor=#1](1,1)
   \psset{fillcolor=#2}
   \pswedge(0,0){0.5}{0}{90} \pswedge(1,1){0.5}{-180}{-90}}
^^A  % Counter to change the random seed
 \newcount\InitCounter
^^A  % LPT tile
 \newcommand{\BasicTileLPT}[2]{%
   \InitCounter=\the\time
   \pscustom{\code{%
     rand \the\InitCounter\space sub 2 mod 0 eq \pslbrace
     row column add 2 mod 0 eq \pslbrace}}
   \begin{pspicture}(1,1)\ProtoTileLPT{#1}{#2}\end{pspicture}%
   \pscustom{\code{\psrbrace \pslbrace}}
   \ProtoTileLPT{#2}{#1}%
   \pscustom{\code{%
     \psrbrace ifelse \psrbrace \pslbrace row column add 2 mod 0 eq \pslbrace}}
   \psrotateleft{\ProtoTileLPT{#2}{#1}}\pscustom{\code{\psrbrace \pslbrace}}
   \psrotateleft{\ProtoTileLPT{#1}{#2}}\pscustom{\code{\psrbrace ifelse \
     psrbrace ifelse}}}
 \psboxfill{\BasicTileLPT{red}{yellow}}
 \Tiling{(4,4)}\hfill
 \psset{unit=0.4}\psboxfill{\BasicTileLPT{blue}{cyan}}
 \Tiling{(15,15)}
```

Another classic example is to generate coordinates and numerotation for a grid. Of course, it is possible to do it directly in PSTricks using nested `\multido` commands. It would be clearly easy to program, but, nevertheless, for users who have a little knowledge of PostScript programming, this offer an alternative which is useful for large cases, because on this way it will be notably faster and less computer ressources consuming.

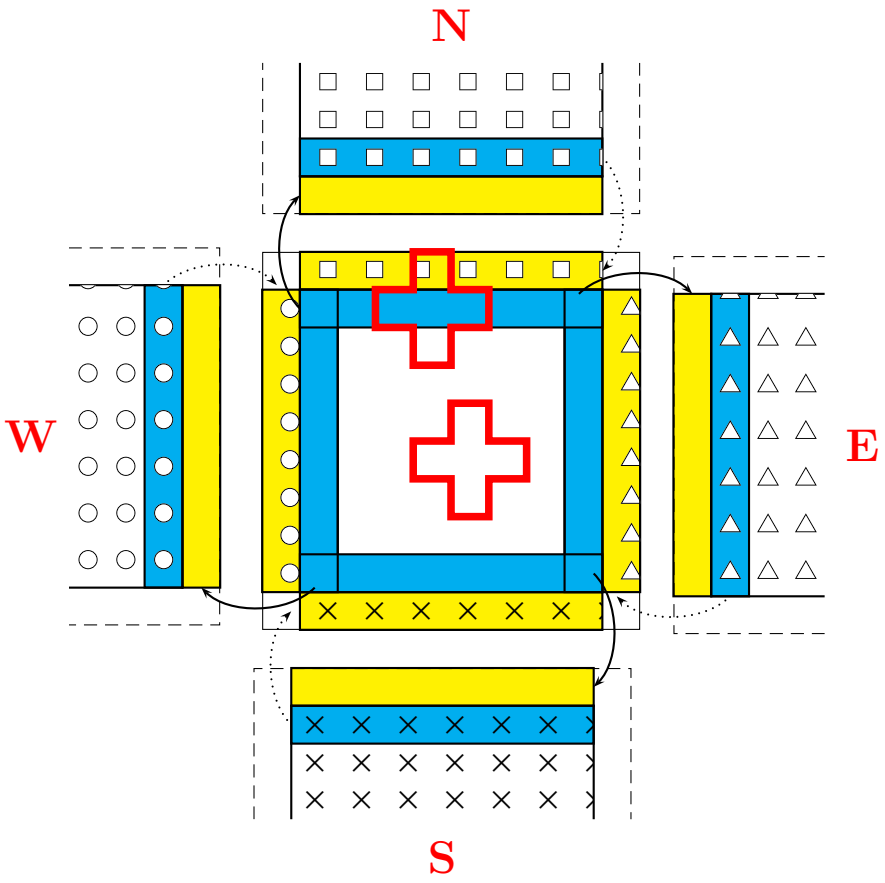
Remember here that the tiling is drawn from left to right, and top to bottom, and note that the PostScript variable `x2` give the total number of columns.




```
^^A  % \Escape will be the \ character
 {\catcode`\!=0\catcode`\\=11!gdef!Escape{\}}
 \newcommand{\ProtoTile}{%
  \Square\pscustom{%
    \moveto(-0.9,0.75) % In PSTricks units
    \code{ /Times-Italic findfont 8 scalefont setfont
      (\Escape() show row 3 string cvs show (,) show
      column 3 string cvs show (\Escape)) show}
    \moveto(-0.5,0.25) % In PSTricks units
    \code{ /Times-Bold findfont 18 scalefont setfont
      1 0 0 setrgbcolor % Red color
      /center {dup stringwidth pop 2 div neg 0 rmoveto} def
      row 1 sub x2 mul column add 3 string cvs center show}}}
 \psboxfill{\ProtoTile}
 \Tiling{(6,4)}
```

4.2 A complete example: the Poisson equation

To finish, we will show a complete real example, a drawing to explain the method used to solve the Poisson equation by a domain decomposition method, adapted to distributed memory computers. The objective is to show the communications required between processes and the position of the data to exchange. This code also show some useful and powerful technics for PSTricks programming (look specially at the way some higher level macros are defined, and how the same object is used to draw the four neighbors).




```
\newcommand{\Pattern}[1]{%
  \begin{pspicture}(-0.25,-0.25)(0.25,0.25)\rput{*0}{\psdot[
    dotstyle=#1]}
  \end{pspicture}}
\newcommand{\West}{\Pattern{o}}   \newcommand{\South}{\Pattern{x
  }}
\newcommand{\Central}{\Pattern{+}}\newcommand{\North}{\Pattern{
  square}}
\newcommand{\East}{\Pattern{triangle}}
\newcommand{\Cross}{%
  \pspolygon[unit=0.5,linewidth=0.2,linecolor=red](0,0)(0,1)(1,1)
    (1,2)(2,2)(2,1)
            (3,1)(3,0)(2,0)(2,-1)(1,-1)(1,0)}
\newcommand{\StylePosition}[1]{\LARGE\textcolor{red}{\textbf
  {#1}}}
\newcommand{\SubDomain}[4]{%
  \psboxfill{#4}
  \begin{psclip}{\psframe[linestyle=none]#1}
    \psframe[linestyle=#3](5,5)\psframe[fillstyle=boxfill]#2
  \end{psclip}}
\newcommand{\SendArea}[1]{\psframe[fillstyle=solid,fillcolor=cyan
  ]#1}
\newcommand{\ReceiveData}[2]{%
```

```
    \psboxfill{#2}
    \psframe[fillstyle=solid,fillcolor=yellow,addfillstyle=boxfill
      ]#1}
  \newcommand{\Neighbor}[2]{%
    \begin{pspicture}(5,5)
      \rput{*0}(2.5,2.5){\StylePosition{#1}}
      \ReceiveData{(0.5,0)(4.5,0.5)}{\Central}\SendArea{(0.5,0.5)
        (4.5,1)}
      \SubDomain{(5,2)}{(0.5,0.5)(4.5,3)}{dashed}{#2}%
^^A       % Receive and send arrows
      \pcarc[arcangle=45,arrows=->](0.5,-1.25)(0.5,0.25)
      \pcarc[arcangle=45,arrows=->,linestyle=dotted,dotsep=2pt
        ](4.5,0.75)(4.5,-0.75)
    \end{pspicture}}
  \psset{dimen=middle,dotscale=2,fillloopadd=2}
  \begin{pspicture}(-5.7,-5.7)(5.7,5.7)
^^A     % Central domain
    \rput(0,0){%
      \begin{pspicture}(5,5)
^^A         % Receive from West, East, North and South
        \ReceiveData{(0,0.5)(0.5,4.5)}{\West} \ReceiveData
          {(4.5,0.5)(5,4.5)}{\East}
        \ReceiveData{(0.5,4.5)(4.5,5)}{\North}\ReceiveData{(0.5,0)
          (4.5,0.5)}{\South}
^^A         % send area for West, East, North and South
        \SendArea{(0.5,0.5)(1,4.5)} \SendArea{(4,0.5)(4.5,4.5)}
        \SendArea{(0.5,0.5)(4.5,1)} \SendArea{(0.5,4)(4.5,4.5)}
^^A         % Central domain
        \SubDomain{(5,5)}{(0.5,0.5)(4.5,4.5)}{solid}{\Central}
^^A         % Redraw overlapped linesY
        \psline(1,0.5)(1,4.5)  \psline(4,0.5)(4,4.5)
^^A         % Two crossesY
        \rput(1.5,4){\Cross}  \rput(2,2){\Cross}
      \end{pspicture}}
^^A     % The four neighborsY
    \rput(0,5.5){\Neighbor{N}{\North}}     \rput{-90}(5.5,0){\
      Neighbor{E}{\East}}
    \rput{90}(-5.5,0){\Neighbor{W}{\West}} \rput{180}(0,-5.5){\
      Neighbor{S}{\South}}
  \end{pspicture}
```

Bibliography

References

[1] Adobe, Systems Incorporated, *PostScript Language Reference Manual*, Addison-Wesley, 2 edition, 1995.

[2] Piotr Bolek, METAPOST and patterns, *TUGboat*, Volume 19, Number 3, pages 276–283, September 1998, `graphics/metapost/macros/mpattern`.

[3] Emmanuel Chailloux, Guy Cousineau and Ascánder Suárez, Programmation fonctionnelle de graphismes pour la production d'illustrations techniques, *Technique et science informatique*, Volume 15, Number 7, pages 977–1007, 1996 (in french).

[4] André Deledicq, *Le monde des pavages*, ACL Éditions, 1997 (in french).

[5] Philippe Esperet and Denis Girou, Coloriage du pavage dit de Truchet, Cahiers GUTenberg, Number 31, pages 5–18, December 1998 (in french).

[6] Denis Girou, Présentation de PSTricks, *Cahiers GUTenberg*, Number 16, pages 21–70, February 1994 (in french).

[7] Michel Goossens, Sebastian Rahtz and Frank Mittelbach, *The LaTeX Graphics Companion*, Addison-Wesley, 2005.

[8] Branko Grünbaum and Geoffrey Shephard, *Tilings and Patterns*, Freeman and Company, 1987.

[9] Alan Hoenig, *TeX Unbound: LaTeX & TeX Strategies, Fonts, Graphics, and More*, Oxford University Press, 1997.

[10] Kristoffer H. Rose and Ross Moore, XY-pic. Pattern and Tile extension, available from CTAN, 1991-1998, `macros/generic/diagrams/xypic`.

[11] Kees van der Laan, Paradigms: Just a little bit of PostScript, *MAPS*, Volume 17, pages 137–150, 1996.

[12] Kees van der Laan, Tiling in PostScript and METAFONT – Escher's wink, *MAPS*, Volume 19, Number 2, pages 39–67, 1997.

[13] Timothy Van Zandt, PSTricks. PostScript macros for Generic TeX, available from CTAN, 1993, `graphics/pstricks`.

[14] Timothy Van Zandt and Denis Girou, Inside PSTricks, *TUGboat*, Volume 15, Number 3, pages 239–246, September 1994.

[15] Herbert Voß, PSTricks – Graphics for TeX and LaTeX, DANTE/Lehmanns, 4th ed., 2007.

[16] Hao Wang, Games, Logic and Computers, *Scientific American*, pages 98–106, November 1965.

5 Driver file

The next bit of code contains the documentation driver file for TeX, i.e., the file that will produce the documentation you are currently reading. It will be extracted from this file by the `docstrip` program.

```
1 ⟨*driver⟩
2 \documentclass{ltxdoc}
3 \GetFileInfo{pst-fill.dtx}
4 %
5 \usepackage[T1]{fontenc}
6 \usepackage{lmodern}                % For PDF
```

```
\usepackage{graphicx}               % 'graphicx' LaTeX standard package
\usepackage{showexpl}
\usepackage{mflogo}                 % For the MetaFont and MetaPost logos
\input{random.tex}                  % Random macros from Donald Arseneau
\usepackage{url}                    % URLs convenient typesetting
\usepackage{multido}                % General loop macro
\usepackage[dvipsnames]{pstricks}   % PSTricks with the 'color' extension
\usepackage{pst-text}               % PSTricks package for character path
\usepackage{pst-grad}               % PSTricks package for gradient filling
\usepackage{pst-node}               % PSTricks package for nodes
\usepackage[tiling]{pst-fill}       % PSTricks package for filling/tiling
%
\AtBeginDocument{%
%  \OnlyDescription % comment out for implementation details
  \EnableCrossrefs
  \CodelineIndex
  \RecordChanges}
\AtEndDocument{%
  \PrintIndex
  \setcounter{IndexColumns}{1}
  \PrintChanges}
\hbadness=7000             % Over and under full box warnings
\hfuzz=3pt
\begin{document}
  \DocInput{pst-fill.dtx}
\end{document}
⟨/driver⟩
```

6 pst-fill LaTeX wrapper

```
⟨*latex − wrapper⟩
\RequirePackage{pstricks}
\ProvidesPackage{pst-fill}[2005/09/13 package wrapper for
  pst-fill.tex (hv)]
\DeclareOption{tiling}{\def\PstTiling{true}}
\ProcessOptions\relax
\input{pst-fill.tex}
\ProvidesFile{pst-fill.tex}
  [\filedate\space v\fileversion\space 'PST-fill' (tvz,dg)]
⟨/latex − wrapper⟩
```

7 Pst-Fill Package code

```
⟨*pst − fill⟩
```

7.1 Preamble

Who we are.

```
\def\fileversion{1.01}
```

```
\def\filedate{2007/03/10}
\message{‘PST-Fill’ v\fileversion, \filedate\space (tvz,dg,hv)}
\csname PSTboxfillLoaded\endcsname
\let\PSTboxfillLoaded\endinput
```

Require the main PSTricks package.

```
\ifx\PSTricksLoaded\endinput\else\input pstricks.tex\fi
```

interface to the extended ‘keyval’ package.

```
\ifx\PSTXKeyLoaded\endinput\else\input pst-xkey\fi
%
```

Catcodes changes and defining the family name for xkeyval.

```
\edef\PstAtCode{\the\catcode‘\@}\catcode‘\@=11\relax

\pst@addfams{pst-fill}
%
```

7.2 The size of the box

pst@@boxfillsize

```
%
\def\pst@@boxfillsize#1(#2,#3)#4(#5,#6)#7(#8\@nil{%
  \begingroup
    \ifx\@empty#7\relax
      \pst@dima\z@
      \pst@dimb\z@
      \pssetxlength\pst@dimc{#2}%
      \pssetylength\pst@dimd{#3}%
    \else
      \pssetxlength\pst@dima{#2}%
      \pssetylength\pst@dimb{#3}%
      \pssetxlength\pst@dimc{#5}%
      \pssetylength\pst@dimd{#6}%
    \fi
    \xdef\pst@tempg{%
      \pst@dima=\number\pst@dima sp
      \pst@dimb=\number\pst@dimb sp
      \pst@dimc=\number\pst@dimc sp
      \pst@dimd=\number\pst@dimd sp }%
  \endgroup
  \let\psk@boxfillsize\pst@tempg}
```

7.3 Definition of the parameters

```
\define@key[psset]{pst-fill}{boxfillsize}{%
  \def\pst@tempg{#1}\def\pst@temph{auto}%
  \ifx\pst@tempg\pst@temph
    \let\psk@boxfillsize\relax
```

```
  \else
    \pst@@boxfillsize#1(\z@,\z@)\@empty(\z@,\z@)(\@nil
  \fi}
\psset{boxfillsize={(-15cm,-15cm)(15cm,15cm)}}
\define@key[psset]{pst-fill}{boxfillcolor}{\pst@getcolor{#1}\psboxfillcolor}
\psset{boxfillcolor=black}% hv
\define@key[psset]{pst-fill}{boxfillangle}{\pst@getangle{#1}\psk@boxfillangle}
\psset{boxfillangle=0}
\define@key[psset]{pst-fill}{fillsepx}{%
  \pst@getlength{#1}\psk@fillsepx}
\define@key[psset]{pst-fill}{fillsepy}{%
  \pst@getlength{#1}\psk@fillsepy}
\define@key[psset]{pst-fill}{fillsep}{%
  \pst@getlength{#1}\psk@fillsepx%
  \let\psk@fillsepy\psk@fillsepx}
\psset{fillsep=2pt}

\ifx\PstTiling\@undefined
  \define@key[psset]{pst-fill}{fillcycle}{\pst@getint{#1}\psk@fillcycle}
  \psset{fillcycle=0}
\else
  \define@key[psset]{pst-fill}{fillangle}{\pst@getangle{#1}\psk@boxfillangle}
  \define@key[psset]{pst-fill}{fillsize}{%
      \def\pst@tempg{#1}\def\pst@temph{auto}%
      \ifx\pst@tempg\pst@temph\let\psk@boxfillsize\relax
      \else\pst@@boxfillsize#1(\z@,\z@)\@empty(\z@,\z@)(\@nil\fi}
  \psset{fillsep=0,fillsize=auto}
  \define@key[psset]{pst-fill}{fillcyclex}{\pst@getint{#1}\psk@fillcyclex}
  \define@key[psset]{pst-fill}{fillcycley}{\pst@getint{#1}\psk@fillcycley}
  \define@key[psset]{pst-fill}{fillcycle}{%
    \pst@getint{#1}\psk@fillcyclex\let\psk@fillcycley\psk@fillcyclex}
  \psset{fillcycle=0}
  \define@key[psset]{pst-fill}{fillmovex}{\pst@getlength{#1}\psk@fillmovex}
  \define@key[psset]{pst-fill}{fillmovey}{\pst@getlength{#1}\psk@fillmovey}
  \define@key[psset]{pst-fill}{fillmove}{%
      \pst@getlength{#1}\psk@fillmovex\let\psk@fillmovey\psk@fillmovex}
  \psset{fillmove=0pt}
  \define@key[psset]{pst-fill}{fillloopaddx}{\pst@getint{#1}\psk@fillloopaddx}
  \define@key[psset]{pst-fill}{fillloopaddy}{\pst@getint{#1}\psk@fillloopaddy}
  \define@key[psset]{pst-fill}{fillloopadd}{%
    \pst@getint{#1}\psk@fillloopaddx\let\psk@fillloopaddy\psk@fillloopaddx}
  \psset{fillloopadd=0}
% For debugging (to debug, set PstDebug=1)
% we now use the one from pstricks to prevent a clash with package
% pstricks                          2004-06-22
%%    \define@key[psset]{pst-fill}{PstDebug}{\pst@getint{#1}\psk@PstDebug}
    \psset{PstDebug=0}
\fi
% DG addition end
```

7.4 Definition of the fill box

psboxfill

```
\newbox\pst@fillbox
\def\psboxfill{\pst@killglue\pst@makebox\psboxfill@i}
\def\psboxfill@i{\setbox\pst@fillbox\box\pst@hbox\ignorespaces}
```

7.5 The main macros

psfs@boxfill

```
\def\psfs@boxfill{%
  \ifvoid\pst@fillbox
    \@pstrickserr{Fill box is empty. Use \string\psboxfill\space first.}\@ehpa
  \else
    \ifx\psk@boxfillsize\relax \pst@AutoBoxFill
    \else\pst@ManualBoxFill\fi
  \fi}
```

pst@ManualBoxFill

```
\def\pst@ManualBoxFill{%
  \leavevmode
  \begingroup
    \pst@FlushCode
    \begin@psclip
    \pstVerb{clip}%
    \expandafter\pst@AddFillBox\psk@boxfillsize
    \end@psclip
  \endgroup}
```

pst@FlushCode

```
\def\pst@FlushCode{%
  \pst@Verb{%
    /mtrxc CM def
    CP CP T
    \tx@STV
    \psk@origin
    \psk@swapaxes
    \pst@newpath
    \pst@code
    mtrxc setmatrix
    moveto
    0 setgray}%
  \gdef\pst@code{}}
```

pst@AddFillBox

```
\def\pst@AddFillBox#1 #2 #3 #4 {%
  \begingroup
    \setbox\pst@fillbox=\vbox{%
      \hbox{\unhcopy\pst@fillbox\kern\psk@fillsepx\p@}%
```

```
      \vskip\psk@fillsepy\p@}%
    \psk@boxfillsize
    \pst@cnta=\pst@dimc
    \advance\pst@cnta-\pst@dima
    \divide\pst@cnta\wd\pst@fillbox
    \pst@cntb=\pst@dimd
    \advance\pst@cntb-\pst@dimb
    \pst@dimd=\ht\pst@fillbox
    \divide\pst@cntb\pst@dimd
    \def\pst@tempa{%
      \pst@tempg
      \copy\pst@fillbox
      \advance\pst@cntc\@ne
      \ifnum\pst@cntc<\pst@cntd\expandafter\pst@tempa\fi}%
    \let\pst@tempg\relax
    \pst@cntc-\tw@
    \pst@cntd\pst@cnta
    \setbox\pst@fillbox=\hbox to \z@{%
      \kern\pst@dima
      \kern-\wd\pst@fillbox
      \pst@tempa
      \hss}%
    \pst@cntd\pst@cntb
%% DG modification begin - Dec. 11, 1997 - Patch 2
    \ifx\PstTiling\@undefined
      \ifnum\psk@fillcycle=\z@\pst@ManualFillCycle\fi
    \else
      \ifnum\psk@fillcyclex=\z@\pst@ManualFillCycle\fi
    \fi
%% DG modification end
    \global\setbox\pst@boxg=\vbox to\z@{%
      \offinterlineskip
      \vss
      \pst@tempa
      \vskip\pst@dimb}%
  \endgroup
  \setbox\pst@fillbox\box\pst@boxg
  \pst@rotate\psk@boxfillangle\pst@fillbox
  \box\pst@fillbox}
```

pst@ManualFillCycle

```
\def\pst@ManualFillCycle{%
  \ifx\PstTiling\@undefined
    \pst@cntg=\psk@fillcycle
  \else
    \pst@cntg=\psk@fillcyclex
  \fi
  \pst@dimg=\wd\pst@fillbox
  \ifnum\pst@cntg=\z@
  \else
```

```
  \divide\pst@dimg\pst@cntg
  \fi
  \ifnum\pst@cntg<\z@\pst@cntg=-\pst@cntg\fi
  \advance\pst@cntg\m@ne
  \pst@cnth=\pst@cntg
  \def\pst@tempg{%
    \ifnum\pst@cnth<\pst@cntg\advance\pst@cnth\@ne\else\pst@cnth\z@\fi
    \moveright\pst@cnth\pst@dimg}}
```

Auto box fill: !! Fix dictionary

7.6 The PostScript subroutines

```
%% DG addition begin - Apr. 8, 1997 and Dec. 1997 - Patch 2
\ifx\PstTiling\@undefined
\pst@def{AutoFillCycle}<%
  /c ED
  /n 0 def
  /s {
    /x x w c div n mul add def
    /n n c abs 1 sub lt { n 1 add } { 0 } ifelse def
  } def>

\pst@def{BoxFill}<%
  gsave
    gsave \tx@STV CM grestore dtransform CM idtransform
    abs /h ED abs /w ED
    pathbbox
    h div round 2 add cvi /y2 ED
    w div round 2 add cvi /x2 ED
    h div round 2 sub cvi /y1 ED
    w div round 2 sub cvi /x1 ED
    /y2 y2 y1 sub def
    /x2 x2 x1 sub def
    CP
    y1 h mul sub neg /y1 ED
    x1 w mul sub neg /x1 ED
    clip
    y2 {
      /x x1 def
      s
      x2 {
        save CP x y1
%% patch 4   hv --------------
        \ifx\VTeXversion\undefined
        \else
%%============ mv: 09-10-01 ??? this is likely to be a right change
        neg
%%============
        \fi
```

```
%% end patch 4
T moveto Box restore
        /x x w add def
      } repeat
      /y1 y1 h add def
    } repeat
    % Next line not useful... To see that, suppress clipping (DG)
    CP x y1 T moveto Box
  currentpoint currentfont grestore setfont moveto>
\else
%% DG modification begin - Apr. 8, 1997 and Nov. / Dec. 1997 - Patch 2
\pst@def{AutoFillCycleX}<%
  /cX ED
  /nX 0 def
  /CycleX {
    /x x w cX div nX mul add def
    /nX nX cX abs 1 sub lt { nX 1 add } { 0 } ifelse def
  } def>
\pst@def{AutoFillCycleY}<%
  /cY ED
  /mY 0 def
  /nY 0 def
  /CycleY {
    /y1 y1 h cY div mY mul sub def
    nY cY abs 1 sub lt { /nY nY 1 add def /mY 1 def }
                       { /nY 0 def        /mY cY abs 1 sub neg def } ifelse
  } def>

\pst@def{BoxFill}<%
  gsave
    gsave \tx@STV CM grestore dtransform CM idtransform
    abs /h ED abs /w ED
    pathbbox
    h div round 2 add cvi /y2 ED
    w div round 2 add cvi /x2 ED
    h div round 2 sub cvi /y1 ED
    w div round 2 sub cvi /x1 ED
    /CoefLoopX 0 def
    /CoefLoopY 0 def
    /CoefMoveX 0 def
    /CoefMoveY 0 def
    \psk@boxfillangle\space 0 ne {/CoefLoopX 8 def /CoefLoopY 8 def} if
    \psk@fillcyclex\space 0 ne {/CoefLoopX CoefLoopX 1 add def} if
    \psk@fillcycley\space 0 ne {/CoefLoopY CoefLoopY 1 add def} if
    \psk@fillmovex\space 0 ne
      {/CoefLoopX CoefLoopX 2 add def
       \psk@fillmovex\space 0 gt {/CoefMoveX CoefLoopX def}
                           {/CoefMoveX CoefLoopX neg def} ifelse} if
    \psk@fillmovey\space 0 ne
      {/CoefLoopY CoefLoopY 2 add def
```

```
      \psk@fillmovey\space 0 gt {/CoefMoveY CoefLoopY def}
                           {/CoefMoveY CoefLoopY neg def} ifelse} if
    \psk@fillsepx\space 0 ne {/CoefLoopX CoefLoopX 1 add def} if
    \psk@fillsepy\space 0 ne {/CoefLoopY CoefLoopY 1 add def} if
    /CoefLoopX CoefLoopX \psk@fillloopaddx\space add def
    /CoefLoopY CoefLoopY \psk@fillloopaddy\space add def
    /x2 x2 x1 sub 4 sub CoefLoopX 2 mul add def
    /y2 y2 y1 sub 4 sub CoefLoopY 2 mul add def
%% We must fix the origin of tiling, as it must not vary according other stuff
%% in the page!
    w x1 CoefLoopX add CoefMoveX add mul
      h y1 y2 add 1 sub CoefLoopY sub CoefMoveY sub mul moveto
    CP
    y1 h mul sub neg /y1 ED
    x1 w mul sub neg /x1 ED
%%  hv 2004-06-22   to prevent clash with pst-gr3d
%%    \psk@PstDebug 0 eq {clip} if
    \Pst@Debug 0 eq {clip} if
%% end hv
    \psk@fillmovex\space \psk@fillmovey
    gsave \tx@STV CM grestore dtransform CM idtransform
    /hmove ED /wmove ED
    /row 0 def
   y2 {
       /row row 1 add def
       /column 0 def
       /x x1 def
       CycleX
       save
       x2 {
          /column column 1 add def
          CycleY
          save CP x y1
%% patch 4   hv --------------
          \ifx\VTeXversion\undefined
          \else
%%============ mv: 09-10-01 ??? this is likely to be a right change
          neg
%%============
          \fi
  T moveto Box restore
          /x x w add def
          0 hmove translate
          } repeat
       restore
       /y1 y1 h add def
       wmove 0 translate
       } repeat
  currentpoint currentfont grestore setfont moveto>
\fi
```

```
\def\pst@AutoBoxFill{%
  \leavevmode
  \begingroup
    \pst@stroke
    \pst@FlushCode
    \pst@Verb{\psk@boxfillangle\space \tx@RotBegin}%
    \pstVerb{\pst@dict /Box \pslbrace end}%
    \ifx\PstTiling\@undefined
    \else
      \ifx\pst@tempa\@undefined % Undefined for instance for \pscharpath
      \else\ifx\pst@tempa\@empty\else
        \def\pst@temph{0}%
        \ifx\pst@tempa\pst@temph
        \else
          \pstVerb{/TR {pop pop currentpoint translate \pst@tempa\space translate } def}%
        \fi
      \fi\fi
    \fi
    \hbox to \z@{\vbox to\z@{\vss\copy\pst@fillbox\vskip-\dp\pst@fillbox}\hss}%
    \ifx\PstTiling\@undefined
      \pstVerb{%
        tx@Dict begin \psrbrace def
        \ifnum\psk@fillcycle=\z@
          /s {} def
        \else
          \psk@fillcycle \tx@AutoFillCycle
        \fi
        \pst@number{\wd\pst@fillbox}%
        \psk@fillsepx\space add
        \pst@number{\ht\pst@fillbox}%
        \pst@number{\dp\pst@fillbox}%
        \psk@fillsepy\space add add
        \tx@BoxFill
        end}%
      \else
      \pstVerb{%
        tx@Dict begin \psrbrace def
        \ifnum\psk@fillcyclex=\z@
          /CycleX {} def
        \else
          \psk@fillcyclex\space \tx@AutoFillCycleX
        \fi
        \ifnum\psk@fillcycley=\z@
          /CycleY {} def
        \else
          \psk@fillcycley\space \tx@AutoFillCycleY
        \fi
        \pst@number{\wd\pst@fillbox}%
        \psk@fillsepx\space add
        \pst@number{\ht\pst@fillbox}%
```

```
        \pst@number{\dp\pst@fillbox}%
        \psk@fillsepy\space add add
        \tx@BoxFill
        end}%
    \fi
    \pst@Verb{\tx@RotEnd}%
  \endgroup}
```

7.7 Closing

Catcodes restoration.

```
\catcode`\@=\PstAtCode\relax
```

⟨/pst − fill⟩

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.

S

T

U

V

W

Change History

v0.93

General: With a \PstTiling macro defined (or "tiling" optional parameter on \usepackage[tiling]pst-fill) there are several add-ons and changes to do 'tiling' rather than 'filling' in "automatic" mode : - we fix the position of the beginning of tiling, - we allow normally the framing of the area as expected, using the

line.... parameters - we define move parameters fillmovex, fillmovey and fillmove, - we define fillcyclex as previous fillcycle parameter, and add the fillcycley and fillcycle (both fillcyclex and fillcycley) ones - we can extend the tiling area using fillloopaddx, fillloopaddy and fillloopadd parameters, - we can debug and see the whole tiling area without clipping using PstDebug parameter, - for names consistancy, we can use fillangle in place of boxfillangle and fillsize in place of boxfillsize, - default value for fillsep is 0 and for fillsize is auto. 1

v0.94

General: With a truemacro defined (or "tiling" optional parameter on \usepackage[tiling]pstfill), this file run exactly as the original boxfill.tex file from Timothy, version 0.94, except a correction in \pst@ManualFillCycle to avoid a division by 0. It's the default. 1

v0.97

General: make it work with VTeX (mv) 1

v0.98

General: delete the Pst@Debug option and use the the one from pstricks to prevent a clash with pst-gr3d (hv) 1

v0.99

General: merge the VTeX and TeX versions (patch 4) (hv) 1

v1.00

General: use pst-xkey for extend keys (hv) 1

v1.01

General: bugfix for incomplete ifx (hv) 1