'pst-fill'
A PSTricks package for filling and tiling areas

Timothy van Zandt[*] (documentation by Denis Girou[†])

December 12, 1997 — Version 97 patch 2
Documentation revised January 28, 1998

Abstract: **'pst-fill'** is a PSTricks (van Zandt, 1993), (Girou, 1994), (van Zandt and Girou, 1994), (Hoenig, 1997), (Goossens, Rahtz, and Mittelbach, 1997) 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

[*]Department of Economics, Princeton University, Princeton, USA — `<tvz@princeton.edu>`.

[†]CNRS/IDRIS — Centre National de la Recherche Scientifique / Institut du Développement et des Ressources en Informatique Scientifique, B.P. 167, 91403 Orsay Cedex, France — `<Denis.Girou@idris.fr>`.

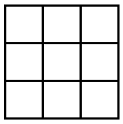
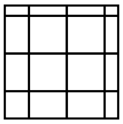
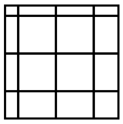
[1]For an extensive presentation of tilings, in their history and usage in many fields, see the reference book (Grünbaum and Shephard, 1987). French readers can also found a lot of explanations and references in (André and Girou, 1998) and specially in (Girou, 1998).

In the TeX world, very few work was done on tilings. You can look at the *tile* extension of the XY-pic package (Rose and Moore, 1991-1998), at the article of Kees van der Laan (van der Laan, 1996, paragraph 7) (the tiling was in fact directly done in PostScript) and at the `METAPOST` program (available on `graphics/metapost/contrib/macros/truchet`) by Denis Roegel for the Truchet contest in 1995 (Esperet and Girou, 1998).

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 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 (Adobe, 1995, 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)}\hspace{3cm}
\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)}\hfill
\Tiling[fillsepy=1mm]{(3,3)}\hfill
\Tiling[fillsep=0.5]{(3,3)}\hfill
\Tiling[fillsep=-0.5]{(3,3)}
```

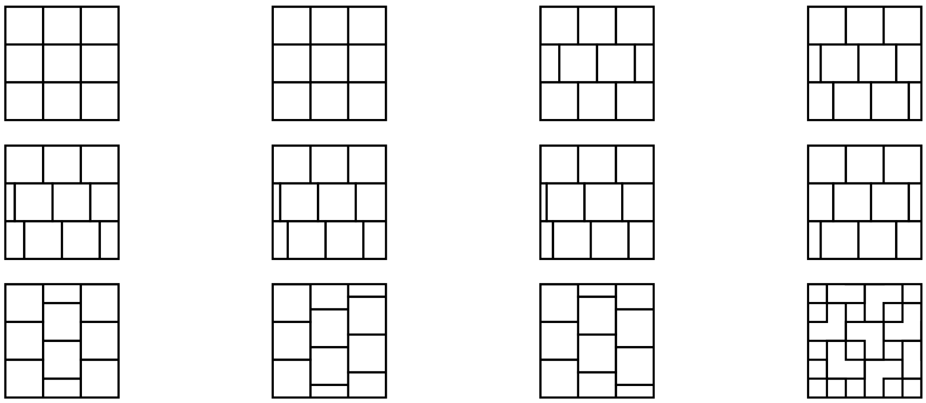
`fillcyclex`[3] (integer): Shift coefficient applied to each row (*Default: 0*).

`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}\hfill\TilingA{1}\hfill\TilingA{2}\hfill\TilingA{3}

\vspace{3mm}
\TilingA{4}\hfill\TilingA{5}\hfill\TilingA{6}\hfill\TilingA{-3}

\vspace{3mm}
\Tiling[fillcycley=2]{(3,3)}\hfill
\Tiling[fillcycley=3]{(3,3)}\hfill
\Tiling[fillcycley=-3]{(3,3)}\hfill
\Tiling[fillcycle=2]{(3,3)}\hfill
```



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

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

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

`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)}\hfill
\Tiling[fillmovey=0.5]{(3,3)}\hfill
\Tiling[fillmove=0.5]{(3,3)}\hfill
\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*).

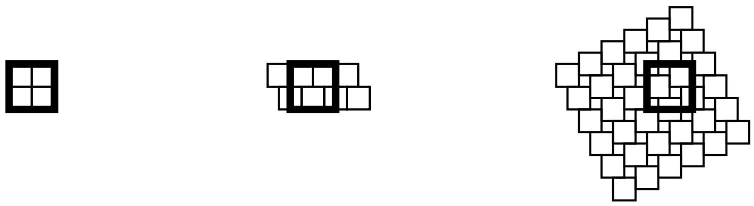
These parameters are only useful in special circonstances, as for complex patterns when the size of the rectangular box used to tile the area doesn't correspond to the pattern itself (see an example in Figure 3.1) and also sometimes when the size of the pattern is not a divisor of the size of the area 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}
\vspace{0.8cm}
\Tiling{(2,2)}\hspace{3cm}
\Tiling[fillcyclex=2]{(2,2)}\hspace{4cm}
\Tiling[fillmove=0.5]{(2,2)}
```



3 Examples

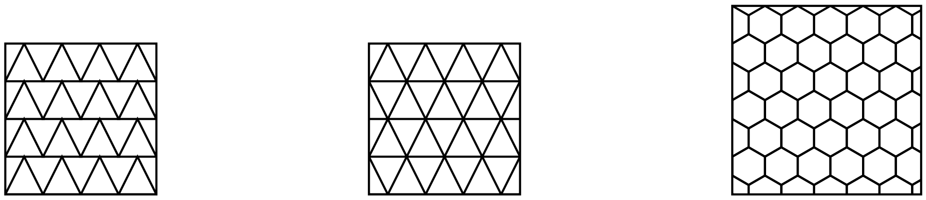
In fact this unique `\psboxfill` macro allow a lot a variations and different usages. We will try here to demonstrate this.

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.

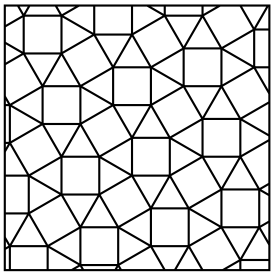
Here we give four other Archimedian tilings (those built with only some regular polygons) among the twelve existing, first discovered completely by Johanes Kepler at the beginning of 17th century (Grünbaum and Shephard, 1987), the two other *regular* ones with the tiling by squares, formed by a unique regular polygon, and two other formed by two different regular polygons.

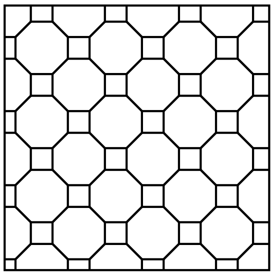
```
\newcommand{\Triangle}{%
  \begin{pspicture}(1,1)
    \pstriangle[dimen=middle](0.5,0)(1,1)
  \end{pspicture}}
\newcommand{\Hexagon}{%
  % sin(60)=0.866
  \begin{pspicture}(0.866,0.75)
    \SpecialCoor
    % Hexagon
    \pspolygon[dimen=middle]
      (0.5;30)(0.5;90)(0.5;150)(0.5;210)(0.5;270)(0.5;330)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

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



```
\newcommand{\ArchimedianA}{%
  % Archimedian tiling 3.4.6.4
  \psset{dimen=middle}
  % 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)
```

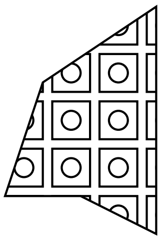
```
  \end{pspicture}}
\newcommand{\ArchimedianB}{%
  % Archimedian tiling 3.12^2
  \psset{dimen=middle,unit=1.5}
  % sin(22.5)=0.3827 ; cos(22.5)=0.9239
  \begin{pspicture}(1.3066,0.6533)
    \SpecialCoor
    % Octogon
    \pspolygon(0.5;22.5)(0.5;67.5)(0.5;112.5)(0.5;157.5)
              (0.5;202.5)(0.5;247.5)(0.5;292.5)(0.5;337.5)
  \end{pspicture}}

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

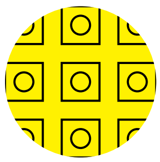




We can of course tile an area arbitrarily defined. And with the `addfillstyle` parameter[4], we can easily mix the `boxfill` style with another one.

```
\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{2cm}
\begin{pspicture}(4,4)
  \pscircle[linestyle=none,fillstyle=solid,fillcolor=yellow,
            addfillstyle=boxfill,fillsep=0.5](2,2){2}
\end{pspicture}
```

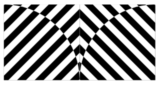


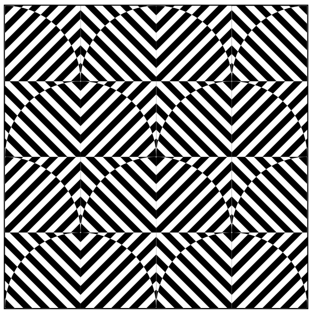


Various effects can be obtained, sometimes complicated ones very easily, as in this example reproduced from one shown by Slavik Jablan in the field of *OpTiles*, inspired by the *Op-art*:

[4]Introduced in PSTricks 97.

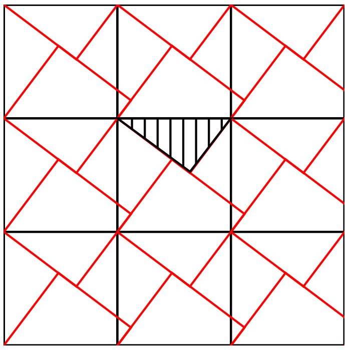
```
\newcommand{\ProtoTile}{%
  \begin{pspicture}(1,1)
    % 1/12=0.08333
    \psset{linestyle=none,linewidth=0,
           hatchwidth=0.08333\psunit,hatchsep=0.08333\psunit}
    \psframe[fillstyle=solid,fillcolor=black,
             addfillstyle=hlines,hatchcolor=white](1,1)
    \pswedge[fillstyle=solid,fillcolor=white,
             addfillstyle=hlines]{1}{0}{90}
  \end{pspicture}}

\newcommand{\BasicTile}{%
  \begin{pspicture}(2,1)
    \rput[lb](0,0){\ProtoTile}
    \rput[lb](1,0){\rotateleft{\ProtoTile}}
  \end{pspicture}}

\ProtoTile\hfill\BasicTile\hfill
\psboxfill{\BasicTile}
\Tiling[fillcyclex=2]{(4,4)}
```





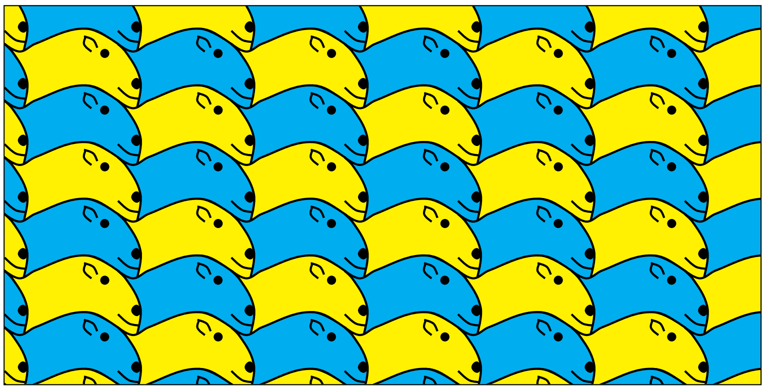
It is also directly possible to surimpose several different tilings. Here is the splendid visual proof of the PYTHAGORE theorem done by the arab mathematician ANNAIRIZI around the year 900, given by superposition of two tilings by squares of different sizes.

```
\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 Andr Deledicq (Deledicq, 1997), 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):

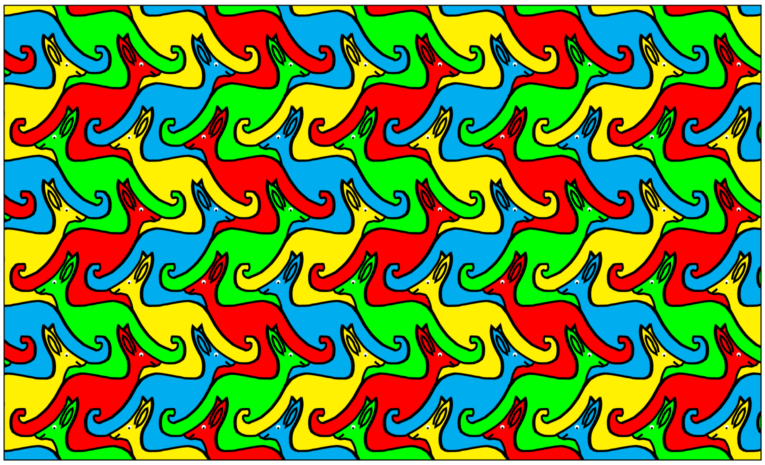
```
\newcommand{\SheepHead}[1]{%
  \begin{pspicture}(3,1.5)
    \pscustom[liftpen=2,fillstyle=solid,fillcolor=#1]{%
      \pscurve(0.5,-0.2)(0.6,0.5)(0.2,1.3)(0,1.5)(0,1.5)
        (0.4,1.3)(0.8,1.5)(2.2,1.9)(3,1.5)(3,1.5)(3.2,1.3)
        (3.6,0.5)(3.4,-0.3)(3,0)(2.2,0.4)(0.5,-0.2)}
    \pscircle*(2.65,1.25){0.12\psunit} % Eye
    \psccurve*(3.5,0.3)(3.35,0.45)(3.5,0.6)(3.6,0.4)% Muzzle
    % Mouth
    \pscurve(3,0.35)(3.3,0.1)(3.6,0.05)
    % 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 (Chailloux, Cousineau, and Suárez, 1996)):

```
\psboxfill{%
  \psset{unit=0.4}
```

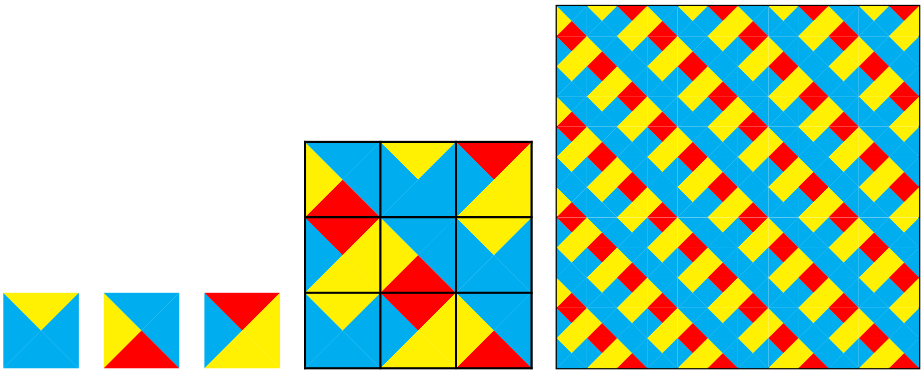
```
  \Kangaroo{yellow}\Kangaroo{red}%
    \Kangaroo{cyan}\Kangaroo{green}%
  \scalebox{-1 1}{%
    \rput(1.235,4.8){%
      \Kangaroo{green}\Kangaroo{cyan}%
        \Kangaroo{red}\Kangaroo{yellow}}}}
\Tiling[fillloopadd=1]{(10,6)}
```



And here a WANG tiling (Wang, 1965), (Grünbaum and Shephard, 1987, 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}}
```

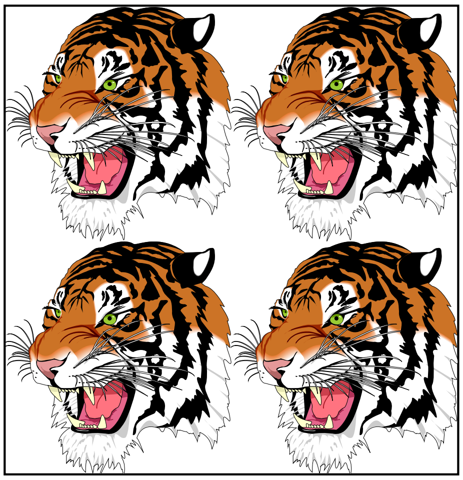
```
\WangTileA\hfill\WangTileB\hfill\WangTileC\hfill
\WangTiles[{\psgrid[subgriddiv=0,gridlabels=0](3,3)}]\hfill
\psset{unit=0.4}
\psboxfill{\WangTiles}
\Tiling{(12,12)}
```



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}(0.1,1)(0.1,4.1)
    \includegraphics[bb=17 176 560 74,width=3cm]{images/tiger}
  \end{pspicture}}
\Tiling{(6,6.2)}
```



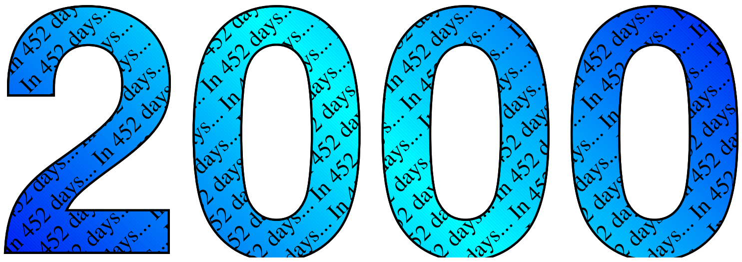
Nevertheless, there are some special files for which the *automatic* mode doesn't work, specially for some files obtained by a screen dump, as in the next example, where a picture was reduced before it conversion in the *Encapsulated PostScript* format by a screen dump utility. In this case, usage of 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{images/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 In 452 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){\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}%
  \scalebox{-1 1}{%
    \rput(1.235,4.8){%
      \Kangaroo{LemonChiffon}\Kangaroo{LightBlue}%
        \Kangaroo{PaleGreen}\Kangaroo{PeachPuff}}}}
% 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` (Girou, 1993-1997) (see fig. 1).

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, as explained in (Girou, 1998)[5].

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

[5] For description of the context, history and references about Sbastien Truchet and this tiling, see (André and Girou, 1998) and specially (André, 1998), (Esperet and Girou, 1998) and (Girou, 1998) .

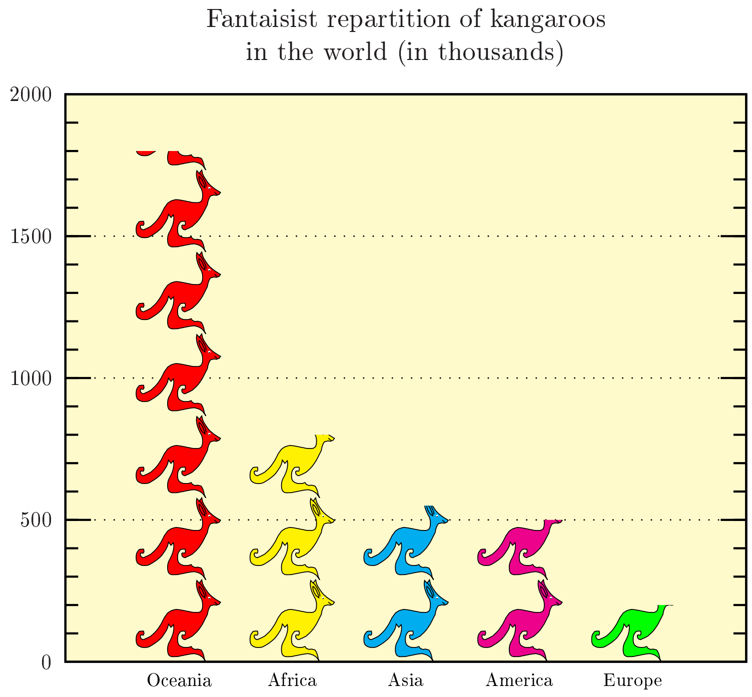
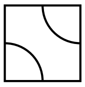
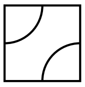


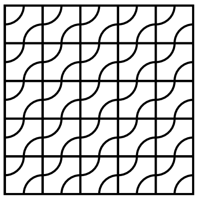
Figure 1: Bar chart generated by PstChart, with bars filled by patterns

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...

```
% 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}}

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

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



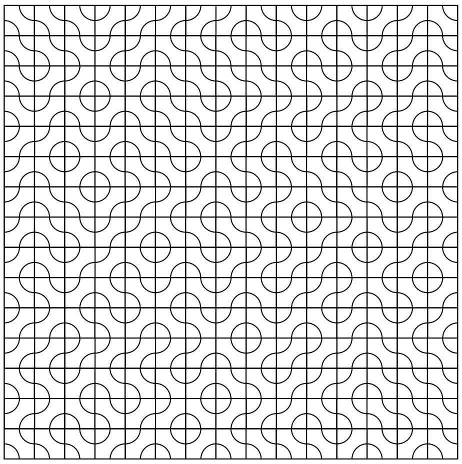




But, for simple cases, there is a solution to this problem using a mixture of PSTricks and PostScript programming. Here the PSTricks construction `\pscustom{\code{...}}` allow to insert PostScript code inside the LaTeX+PSTricks one.

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 `\{` and `\}` are two PSTricks macros to define and be able to insert the `{` and `}` characters.

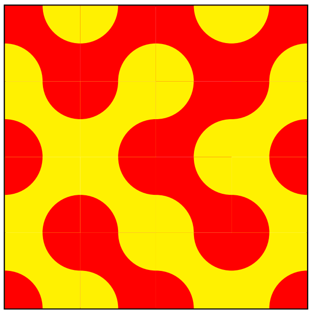
```
% LPT prototile
\newcommand{\ProtoTileLPT}{%
  \psset{dimen=middle}
  \psframe(1,1)
  \psarc(0,0){0.5}{0}{90}
  \psarc(1,1){0.5}{-180}{-90}}

% Counter to change the random seed
\newcount\InitCounter

% 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}}
  \rotateleft{\ProtoTileLPT}%
  \pscustom{\code{\psrbrace ifelse}}}

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

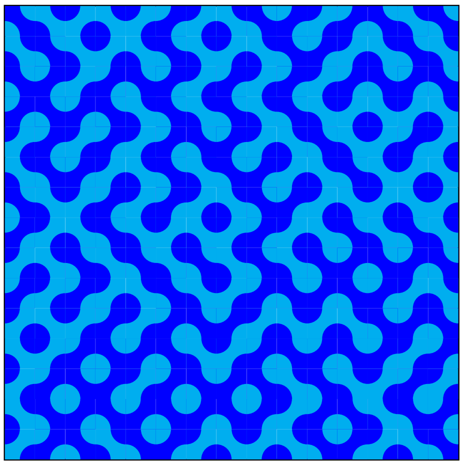


Using the very surprising fact (see (Esperet and Girou, 1998)) that coloration of these tiles do not depend of their neighbors (even if it is difficult to believe as the opposite seems obvious!) but only of the parity of the value of row and column positions, we can directly program in the same way a 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.

```
% 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}}

% Counter to change the random seed
\newcount\InitCounter

% 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}}
  \rotateleft{\ProtoTileLPT{#2}{#1}}%
  \pscustom{\code{\psrbrace \pslbrace}}
  \rotateleft{\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.

```
% \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)}
```

(1,1) 1	(1,2) 2	(1,3) 3	(1,4) 4	(1,5) 5	(1,6) 6
(2,1) 7	(2,2) 8	(2,3) 9	(2,4) 10	(2,5) 11	(2,6) 12
(3,1) 13	(3,2) 14	(3,3) 15	(3,4) 16	(3,5) 17	(3,6) 18
(4,1) 19	(4,2) 20	(4,3) 21	(4,4) 22	(4,5) 23	(4,6) 24

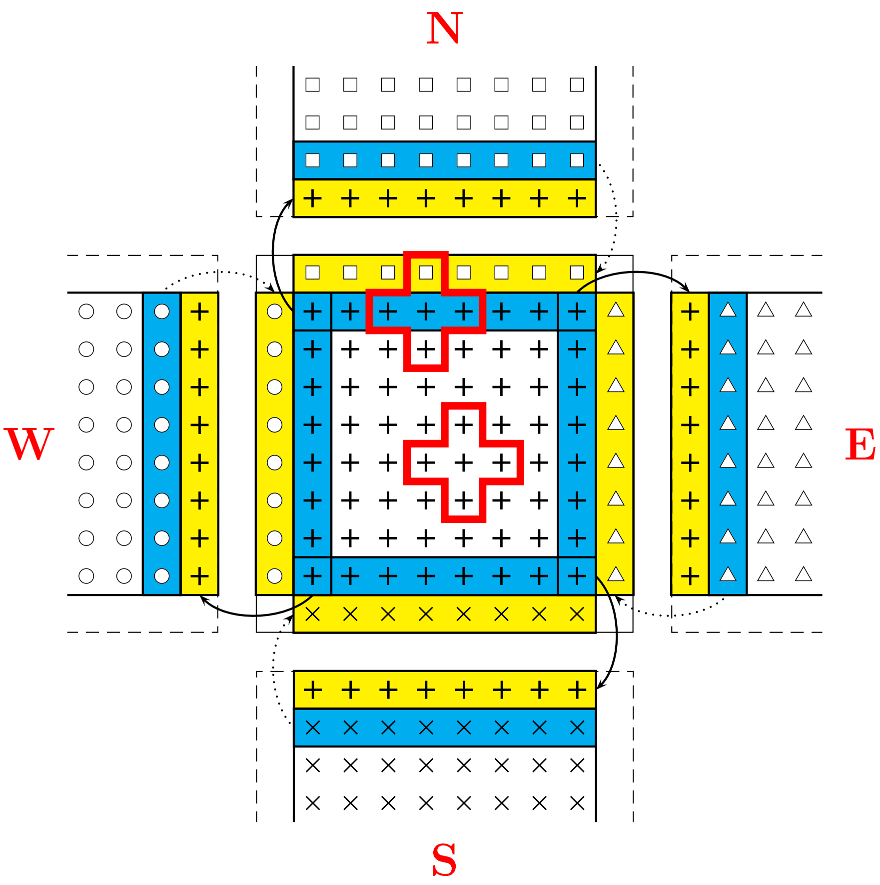
4.2 A complete example : the Poisson equation

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}%
    % 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)
  % Central domain
  \rput(0,0){%
    \begin{pspicture}(5,5)
      % 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}
      % 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)}
      % Central domain
      \SubDomain{(5,5)}{(0.5,0.5)(4.5,4.5)}{solid}{\Central}
      % Redraw overlapped lines
      \psline(1,0.5)(1,4.5)
      \psline(4,0.5)(4,4.5)
```

```
      % Two crosses
      \rput(1.5,4){\Cross}
      \rput(2,2){\Cross}
    \end{pspicture}}
  % The four neighbors
  \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}
```



References

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

[André, 1998] André, Jacques. "Truchet et l'algorithmique". *Éditions des Cahiers GUTenberg* pages ??–??, To appear.

[André and Girou, 1998] André, Jacques and D. Girou. *Sébastien Truchet — Les pavages (Temporary title)*. Éditions des Cahiers Gutenberg, To appear.

[Chailloux, Cousineau, and Suárez, 1996] Chailloux, Emmanuel, G. Cousineau, and A. Suárez. "Programmation fonctionnelle de graphismes pour la production d'illustrations techniques". *Technique et science informatique* **15**(7), 977–1007, 1996.

[Deledicq, 1997] Deledicq, André. *Le monde des pavages*. ACL Éditions, 1997.

[Esperet and Girou, 1998] Esperet, Philippe and D. Girou. "La coloration des pavages. L'exemple du pavage de Truchet". *Éditions des Cahiers GUTenberg* pages ??–??, To appear.

[Girou, 1993-1997] Girou, Denis. "PstChart. Business charts in (LA)TEX + PostScript with PSTricks". `http://www.tug.org/applications/PSTricks/PstChart`, 1993-1998.

[Girou, 1994] Girou, Denis. "Présentation de PSTricks". *Cahiers GUTenberg* **16**, 21–70, 1994.

[Girou, 1998] Girou, Denis. "Les pavages, de l'art à la science. L'exemple du pavage de Truchet". *Éditions des Cahiers GUTenberg* pages ??–??, To appear.

[Goossens, Rahtz, and Mittelbach, 1997] Goossens, Michel, S. Rahtz, and F. Mittelbach. *The LATEX Graphics Companion*. Addison-Wesley, 1997.

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

[Hoenig, 1997] Hoenig, Alan. *TEX Unbound: LATEX & TEX Strategies, Fonts, Graphics, and More*. Oxford University Press, 1997.

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

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

[van Zandt, 1993] van Zandt, Timothy. "PSTricks. PostScript macros for Generic TEX". Available from CTAN, `graphics/pstricks`, 1993.

[van Zandt and Girou, 1994] van Zandt, Timothy and D. Girou. "Inside PSTricks". *TUGboat* **15**(3), 239–246, 1994.

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