

Graphics Applications

pst-fill — a PSTricks package for filling and tiling areas

Denis Girou

Abstract

pst-fill is a PSTricks (van Zandt, 1993), (Girou, 1994), (van Zandt and Girou, 1994), (Hoenig, 1998), (Goossens, Rahtz, and Mittelbach, 1997) package for simple drawing of various kinds of filling and area tiling. It is also a good example of the great power and flexibility of PSTricks, as it is very short (around 200 lines long) but nevertheless extremely powerful.

The package was written in 1994 by Timothy van Zandt but publicly available only in PSTricks 97 and without any documentation. We describe here version *97 patch 2* of December 12, 1997, which is the original one modified by Denis Girou to manage *tilings* in so-called *automatic* mode. This article serves as both reference manual and user's guide.¹

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

1 Introduction

We use *filling* to describe the operation which consists of filling a defined area by a pattern (or a composition of patterns), and *tiling* as the operation which is like filling, but with control of the starting point (we use the upper left corner), where the pattern is positioned relative to this point. There is an essential difference between the two modes, as without control of the starting point we cannot create the *tilings* (sometimes called *tesselations*) used in many fields of Art and Science².

Tilings are a wide and difficult field of mathematics, and this package is limited to simple ones, mainly *monohedral* tilings with one prototile (which

¹ Great thanks are due to Sebastian Rahtz for his help in correcting my English and of course to Timothy van Zandt for his impressive development of the PSTricks package.

² For an extensive description of tilings, and their history and usage in many fields, see the reference book (Grünbaum and Shephard, 1987). French readers can also find much explanation and reference material in (André and Girou, To appear), and especially in (Girou, To appear).

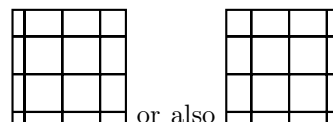
In the \TeX world, very little work has been done on tilings. There is mainly the *tile* extension of the Xy-pic package (Rose and Moore, 1991-1998), the article of Kees van der Laan (van der Laan, 1996, paragraph 7) (the tiling was in fact done directly in PostScript) and the MetaPost program (available in `graphics/metapost/contrib/macros/truchet`) by Denis Roegel for the Truchet contest in 1995 (Esperet and Girou, To appear).

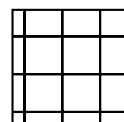
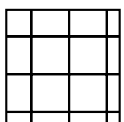
can be composite, see section 3.1). With some experience and wiliness we can do more, and easily obtain quite sophisticated results, but obviously hyperbolic tilings like the famous Escher ones or aperiodic tilings like the Penrose ones are not within the capabilities of this package. For more complex needs, we must use low level and more painfull techniques, with the basic `\multido` and `\multirput` macros.

2 History of the package, and its two different modes

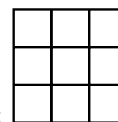
This package was written in 1994 by Timothy van Zandt. Two modes are defined, called respectively *manual* and *automatic*. For both, the pattern is generated on contiguous positions in a large area which includes the region to fill, which is later cut to the required dimensions by a clipping mechanism. In the first mode, the pattern is explicitly inserted in the PostScript output file each time. In the second, the result is the same but with a single insertion of the pattern and a repetition done by PostScript. Control over the starting point was lost, so it allowed only *filling* a region and not to *tiling* it.

The difference between the two modes is shown



here; *filling*:  or also  where, as you can see, the initial position is arbitrary and

depends on the current point, and *tiling*:



It is clear that filling is very restrictive compared to tiling, as the desired effect very often requires the possibility of controlling the starting point. The automatic mode was therefore of limited interest, but unfortunately the *manual* one has the very big disadvantage of requiring very large resources, in disk space and subsequently in printing time. A small tiling can sometimes require several megabytes in *manual* mode! The original package was thus not really usable in practice for tilings.

I modified the code to allow tiling in *automatic* mode, also giving control over the starting point. Most of the time, if some special options are not used, the tiling is done exactly in the region described, which make it faster. There is little reason to use the *manual* mode, apart very special cases where the *automatic* one cannot work, as explained later – currently, we know of 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, simply do not use the *tiling* optional.

Users should be aware that in *tiling* mode, some other changes were introduced. Aliases for some parameter names were defined for consistency (all parameters begin with the `fill` prefix) and some default values which were not well adapted for tilings were changed (`fillsep` is set to 0 and `fillsize` set to `auto`). `fillcycle` was renamed to `fillcyclex`, and the normal behaviour was restored whereby the frame of the area is drawn and all line (`linestyle`, `linecolor`, `doubleline`, etc.) parameters are now active (but not in non *tiling* mode). Some new parameters were introduced to control tiling, described below.

In all the following examples, we always use *tiling* mode.

To do a tiling, we just have 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 be important in some circumstances.

PostScript programmers may be interested to know that, even in *automatic* mode, the iterations of the pattern are managed directly by the PostScript code of the package, which uses only PostScript Level 1 operators. The special ones introduced in Level 2 for drawing patterns (Adobe, 1995, section 4.9) are not used.

First, for convenience, we define a simple `\Tiling` macro, which will simplify our examples:

```
1 \newcommand{\Tiling}[2] [] {%
2   \edef\Temp{#1}%
3   \begin{pspicture}#2
4     \ifx\Temp\empty
5       \psframe[fillstyle=boxfill]#2
6     \else
7       \psframe[fillstyle=boxfill, #1]#2
8     \fi
9   \end{pspicture}}
```

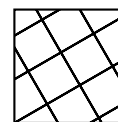
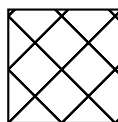
2.1 Parameters

There are 14 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 noticeably larger than the area to cover, to be sure that the defined area will be covered after rotation.

```
1 \newcommand{\Square}{%
2   \begin{pspicture}(1,1)
3     \psframe[dimen=middle](1,1)
4   \end{pspicture}}
5
6 \psset{unit=0.5}
7 \psboxfill{\Square}
8 \Tiling[fillangle=45]{(3,3)}\hspace{3cm}
9 \Tiling[fillangle=-60]{(3,3)}
```



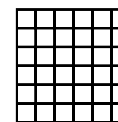
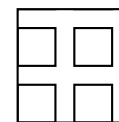
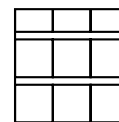
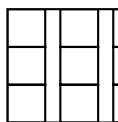
`fillsepx` (real|dim): value of the horizontal separation between consecutive patterns (*Default: 0 for tilings³, 2pt otherwise*).

`fillsepy` (real|dim): value of the vertical separation between consecutive patterns (*Default: 0 for tilings³, 2pt otherwise*).

`fillsep` (real|dim): value of horizontal and vertical separations between consecutive patterns (*Default: 0 for tilings³, 2pt otherwise*).

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

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



`fillcyclex`⁴ (integer): Shift coefficient applied to each row (*Default: 0*).

`fillcycley`³ (integer): Same thing for columns (*Default: 0*).

`fillcycle`³ (integer): Allow for setting both `fillcyclex` and `fillcycley` to the same value (*Default: 0*).

For instance, if `fillcyclex` is 2, the second row of patterns will be horizontally shifted by a factor

³ This option was added by me. It is not part of the original package and is available only if the `tiling` keyword is used when loading the package.

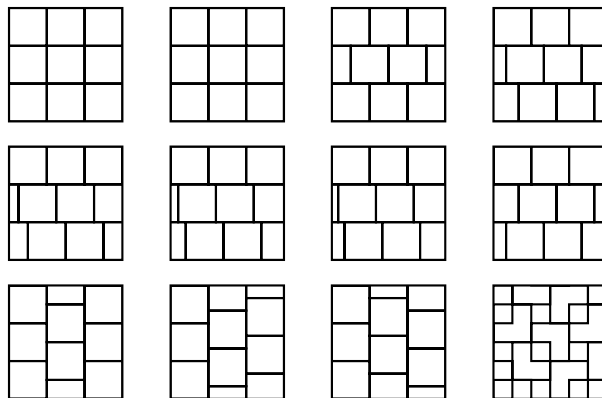
⁴ It was `fillcycle` in the original version.

of $\frac{1}{2} = 0.5$, and by a factor of 0.333 if `fillcyclex` is 3, etc. These values can be negative.

```

1 \psset{unit=0.5}
2 \psboxfill{\Square}
3 \newcommand{\TilingA}[1]
4   {\Tiling[fillcyclex=#1]{(3,3)}}
5
6 \TilingA{0}\hfill
7 \TilingA{1}\hfill
8 \TilingA{2}\hfill
9 \TilingA{3}
10
11 \vspace{3mm}
12 \TilingA{4}\hfill
13 \TilingA{5}\hfill
14 \TilingA{6}\hfill
15 \TilingA{-3}
16
17 \vspace{3mm}
18 \Tiling[fillcyclex=2]{(3,3)}\hfill
19 \Tiling[fillcyclex=3]{(3,3)}\hfill
20 \Tiling[fillcyclex=-3]{(3,3)}\hfill
21 \Tiling[fillcyclex=2]{(3,3)}\hfill

```



`fillmovex`³ (real|dim): value of the horizontal move between consecutive patterns (*Default: 0*).

`fillmoyey`³ (real|dim): value of the vertical move between consecutive patterns (*Default: 0*).

`fillmove`³ (real|dim): value of horizontal and vertical move 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 as that with the `fillcycle?` ones, but this is not true for all values.

```

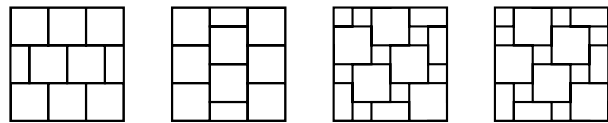
1 \psset{unit=0.5}
2 \psboxfill{\Square}

```

```

3 \Tiling[fillmovex=0.5]{(3,3)}\hfill
4 \Tiling[fillmoyey=0.5]{(3,3)}\hfill
5 \Tiling[fillmove=0.5]{(3,3)}\hfill
6 \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 use up a large amount of computer resources. It's usage is therefore discouraged in favour of *automatic* mode. It only seems useful in special circumstances, when the *automatic* mode fails; only one case is known, when some kinds of EPS files are used, such as the ones produced by partial screen dumps (see 3.2).

`fillloopaddx`³ (integer): number of times the pattern is added on left and right positions (*Default: 0*).

`fillloopaddy`³ (integer): number of times the pattern is added on top and bottom positions (*Default: 0*).

`fillloopadd`³ (integer): number of times the pattern is added on left, right, top and bottom positions (*Default: 0*).

These parameters (exclusively for the *tiling* mode) are only useful in special circumstances, such as in complex patterns when the size of the rectangular box used to tile the area does not correspond to the pattern itself (there is an example in Figure 1) and also sometimes when the size of the pattern is not a divisor of the size of the area to fill and when the number of loop repeats is not properly computed, which can occur.

`PstDebug`³ (integer, 0 or 1): to see the exact tiling done, without clipping (*Default: 0*).

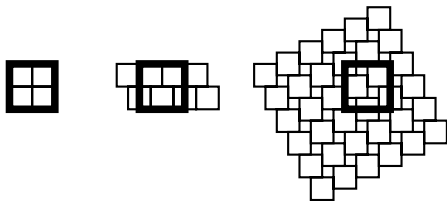
This is mainly useful for debugging or to understand better how the tilings are done. It is implemented only for the *tiling* mode.

```

1 \psset{unit=0.3,PstDebug=1}
2 \psboxfill{\Square}
3 \psset{linewidth=1mm}
4 \vspace*{7mm}
5 \Tiling{(2,2)}\hspace{1cm}

```

```
6 \Tiling[fillcyclex=2]{(2,2)}\hspace{2cm}
7 \Tiling[fillmove=0.5]{(2,2)}
```



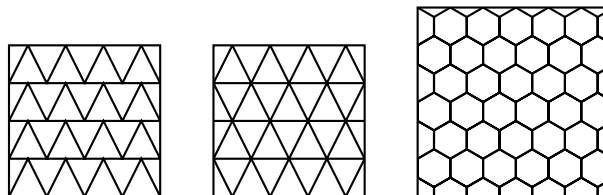
3 Examples

The single `\psboxfill` macro has many variations and different uses. We will try here to demonstrate many of them:

3.1 Kind of tiles

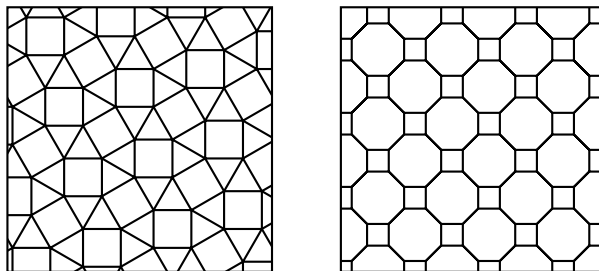
Since we can access all the power of PSTricks macros to define the *tiles (patterns)* used, very complicated ones can be created. Here we give four Archimedean tilings (those built with only some regular polygons) from the eleven known, first discovered completely by Johannes Kepler at the beginning of 17th century (Grünbaum and Shephard, 1987), the two *regular* ones with the tiling by squares, formed by a single regular polygon, and two formed by two different regular polygons.

```
1 \newcommand{\Triangle}{%
2   \begin{pspicture}(1,1)
3     \pstriangle[dimen=middle](0.5,0)(1,1)
4   \end{pspicture}}
5 \newcommand{\Hexagon}{%
6   % sin(60)=0.866
7   \begin{pspicture}(0.866,0.75)
8     \SpecialCoor
9     % Hexagon
10    \pspolygon[dimen=middle]
11      (0.5;30)(0.5;90)(0.5;150)
12      (0.5;210)(0.5;270)(0.5;330)
13  \end{pspicture}}
14
15 \psset{unit=0.5}
16 \psboxfill{\Triangle}
17 \Tiling{(4,4)}\hfill
18 % The two other regular tilings
19 \Tiling[fillcyclex=2]{(4,4)}\hfill
20 \psboxfill{\Hexagon}
21 \Tiling[fillcyclex=2,fillloopaddy=1]{(5,5)}
```



```
1 \newcommand{\ArchimedeanA}{%
2   % Archimedean tiling 3.4.6.4
```

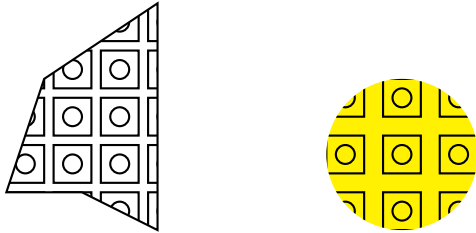
```
3 \psset{dimen=middle}
4 % sin(60)=0.866
5 \begin{pspicture}(1.866,1.866)
6   \psframe(1,1)
7   \psline(1,0)(1.866,0.5)(1,1)
8     (0.5,1.866)(0,1)(-0.866,0.5)
9   \psline(0,0)(0.5,-0.866)
10  \end{pspicture}}
11 \newcommand{\ArchimedeanB}{%
12   % Archimedean tiling 3.12^2
13   \psset{dimen=middle,unit=1.5}
14   % cos(22.5) + sin(22.5) = 1.3066
15   % cos(22.5) - sin(22.5) = 0.6533
16   \begin{pspicture}(1.3066,0.6533)
17     \SpecialCoor
18     % Octagon
19     \pspolygon(0.5;22.5)(0.5;67.5)
20       (0.5;112.5)(0.5;157.5)(0.5;202.5)
21       (0.5;247.5)(0.5;292.5)(0.5;337.5)
22   \end{pspicture}}
23
24 \psset{unit=0.5}
25 \psboxfill{\ArchimedeanA}
26 \Tiling[fillmove=0.5]{(7,7)}\hfill
27 \psboxfill{\ArchimedeanB}
28 \Tiling[fillcyclex=2,fillloopaddy=1]{(7,7)}
```



We can of course tile an arbitrarily defined area; with the `addfillstyle` parameter⁵, we can easily mix the `boxfill` style with another one.

```
1 \psset{unit=0.5,dimn=middle}
2 \psboxfill{%
3   \begin{pspicture}(1,1)
4     \psframe(1,1)
5     \pscicle(0.5,0.5){0.25}
6   \end{pspicture}}
7 \begin{pspicture}(4,6)
8   \pspolygon[fillstyle=boxfill,
9     fillsep=0.25]
10     (0,1)(1,4)(4,6)(4,0)(2,1)
11 \end{pspicture}
12 \hspace{2cm}
13 \begin{pspicture}(4,4)
14   \pscicle[linestyle=none,fillstyle=solid,
15     fillcolor=yellow,fillsep=0.5,
16     addfillstyle=boxfill](2,2){2}
17 \end{pspicture}
```

⁵ Introduced in PSTricks 97.

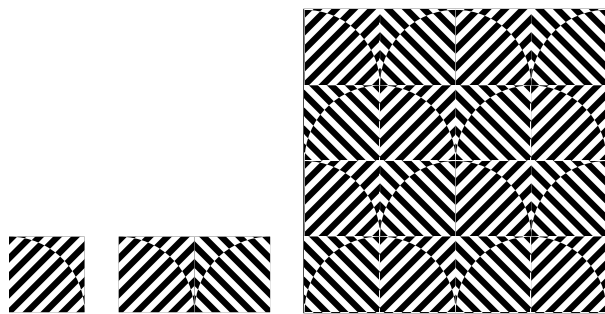


Various effects can be obtained; sometimes complicated ones are surprisingly easy, as in this example reproduced from one by Slavik Jablan in the field of *OpTiles*, inspired by *Op-art*:

```

1 \newcommand{\ProtoTile}{%
2   \begin{pspicture}(1,1)
3     % 1/12=0.08333
4     \psset{linestyle=none,linewidth=0,
5             hatchwidth=0.08333\psunit,
6             hatchsep=0.08333\psunit}
7     \psframe[fillstyle=solid,fillcolor=black,
8             addfillstyle=hlines,
9             hatchcolor=white](1,1)
10    \pswedge[fillstyle=solid,fillcolor=white,
11            addfillstyle=hlines]{1}{0}{90}
12    \end{pspicture}}
13
14 \newcommand{\BasicTile}{%
15   \begin{pspicture}(2,1)
16     \rput[lb](0,0){\ProtoTile}
17     \rput[lb](1,0){\rotateleft{\ProtoTile}}
18   \end{pspicture}}
19
20 \ProtoTile\hfill\BasicTile\hfill
21 \psboxfill{\BasicTile}
22 \Tiling[fillcyclex=2]{(4,4)}

```



It is also possible to superimpose 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.

```

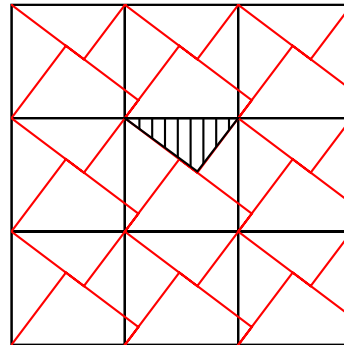
1 \psset{unit=1.5,dimem=middle}
2 \begin{pspicture*}(3,3)
3   \psboxfill{\begin{pspicture}(1,1)
4     \psframe(1,1)
5     \end{pspicture}}
6   \psframe[fillstyle=boxfill](3,3)

```

```

7   \psboxfill{\begin{pspicture}(1,1)
8     \rput{-37}{\psframe[linecolor=red]
9       (0.8,0.8)}
10    \end{pspicture}}
11   \psframe[fillstyle=boxfill](3,4)
12   \pspolygon[fillstyle=hlines,hatchangle=90]
13     (1,2)(1.64,1.53)(2,2)
14 \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 (Deledicq, 1997), Figure 1 shows 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 crystallographer Jevgraf Fedorov at the end of the 19th century).

Figure 2 shows 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 reproduced from an original picture by Raoul Raba and here is a translation into PSTricks from the one drawn by Emmanuel Chailloux and Guy Cousineau for their MLgraph system (Chailloux, Cousineau, and Suárez, 1996)).

And now a Wang tiling (Wang, 1965), (Grünbaum and Shephard, 1987, chapter 11), based on very simple tiles in the form of a square and composed of four colored triangles. Such tilings are simply built with a matching color constraint. Despite its simplicity, it is an important kind of tiling, 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.

```

1 \newcommand{\WangTile}[4]{%
2   \begin{pspicture}(1,1)
3     \pspolygon*[linecolor=#1](0,0)(0,1)(0.5,0.5)
4     \pspolygon*[linecolor=#2](0,1)(1,1)(0.5,0.5)
5     \pspolygon*[linecolor=#3](1,1)(1,0)(0.5,0.5)
6     \pspolygon*[linecolor=#4](1,0)(0,0)(0.5,0.5)
7   \end{pspicture}}
8

```

```

1 \newcommand{\SheepHead}[1]{%
2 \begin{pspicture}(3,1.5)
3   \pscustom[liftpen=2,fillstyle=solid,fillcolor=#1]{%
4     \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)
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)}
6     \pscircle*(2.65,1.25){0.12\psunit}           % Eye
7     \psccurve*(3.5,0.3)(3.35,0.45)(3.5,0.6)(3.6,0.4) % Muzzle
8     \pscurve(3,0.35)(3.3,0.1)(3.6,0.05)         % Mouth
9     \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) % Ear
10  \end{pspicture}}
11
12 \psboxfill{\psset{unit=0.4}\SheepHead{yellow}\SheepHead{cyan}}
13 \Tiling[fillcyclex=2,fillloopadd=1]{(10,5)}

```

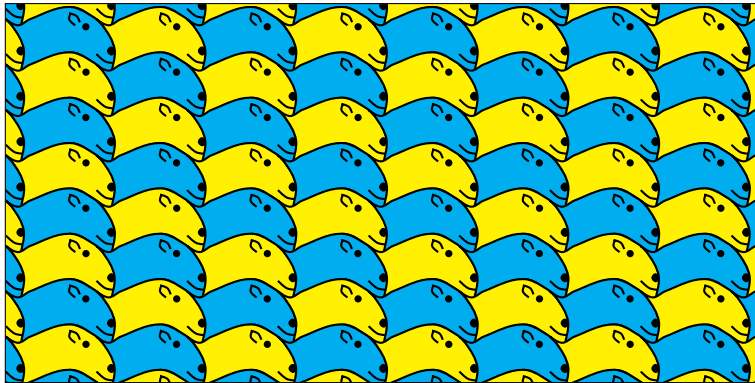


Figure 1: Tiling of $p1$ category

```

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

```

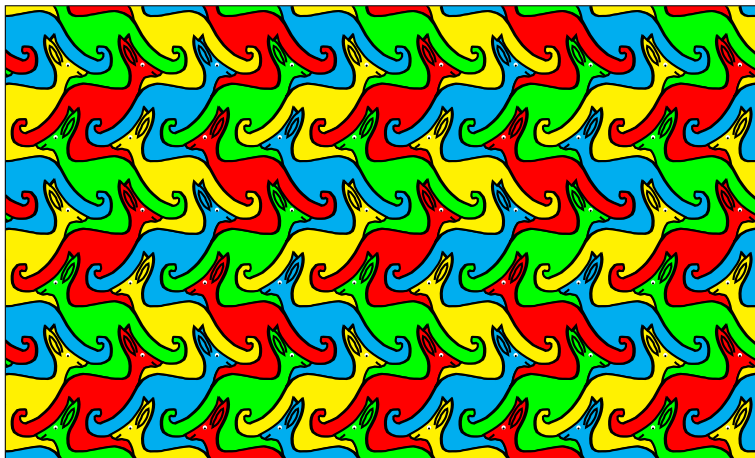
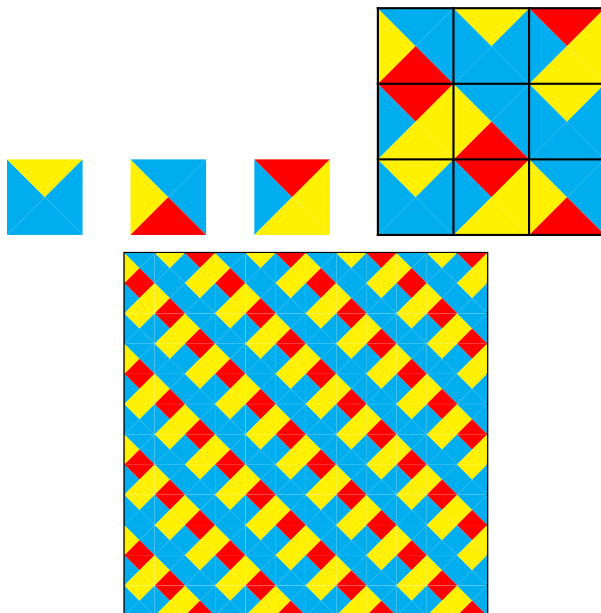


Figure 2: Tiling of pg category

```

9 \newcommand{\WangTileA}{%
10   \WangTile{cyan}{yellow}{cyan}{cyan}}
11 \newcommand{\WangTileB}{%
12   \WangTile{yellow}{cyan}{cyan}{red}}
13 \newcommand{\WangTileC}{%
14   \WangTile{cyan}{red}{yellow}{yellow}}
15
16 \newcommand{\WangTiles}[1] []{%
17   \begin{pspicture}(3,3)
18     \psset{ref=lb}
19     \rput(0,2){\WangTileB}%
20     \rput(1,2){\WangTileA}%
21     \rput(2,2){\WangTileC}%
22     \rput(0,1){\WangTileC}%
23     \rput(1,1){\WangTileB}%
24     \rput(2,1){\WangTileA}%
25     \rput(0,0){\WangTileA}%
26     \rput(1,0){\WangTileC}%
27     \rput(2,0){\WangTileB}
28     #1
29   \end{pspicture}}
30
31 \WangTileA\hfill\WangTileB
32 \hfill\WangTileC\hfill
33 \WangTiles[{\psgrid[subgriddiv=0,
34               gridlabels=0]}(3,3)]
35
36 \vspace{2mm}
37 \psset{unit=0.4}
38 \psboxfill{\WangTiles}
39 \Tiling{(12,12)}

```



3.2 External graphic files

We can fill an arbitrary area with an external PostScript image. We have only, as usual, to worry about the *BoundingBox* definition if there is not one provided or if it is inaccurate, as in the case of the

well known **tiger** picture (part of the Ghostscript distribution).

```

1 \psboxfill{%
2   \raisebox{-1cm}{%
3     \includegraphics[bb=17 176 562 740,
4                       width=3cm]{tiger}}
5 \Tiling{(6,6.2)}

```

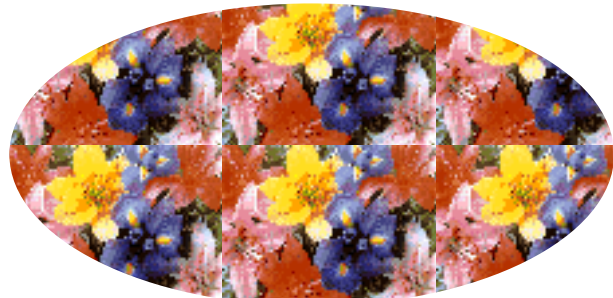


Be warned there are some types of PostScript file for which the *automatic* mode does not work, specifically those produced by a screen dump. This is demonstrated in the next example, where a picture was reduced before conversion to the *Encapsulated PostScript* format by a screen dump utility. In this case, use of the *manual* mode is the only alternative, at the price of 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 too many times, perhaps just actually using a few occurrences as the other ones are clipped away...

```

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

```



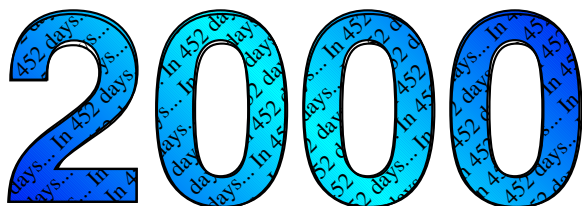
3.3 Tiling of characters

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

```

1 \DeclareFixedFont{\Sf}{T1}{phv}{b}{n}{3.5cm}
2 \DeclareFixedFont{\Rm}{T1}{ptm}{m}{n}{3mm}
3 \psboxfill{\Rm In 452 days...}
4 \begin{pspicture}(8,3)
5   \rput(4,0.2){%
6     \pscharpath[fillstyle=gradient,
7       gradangle=-45,gradmidpoint=0.5,
8       addfillstyle=boxfill,
9       fillangle=45,fillsep=0.7mm]
10    {\rput[b](0,0){\Sf 2000}}
11 \end{pspicture}

```



```

1 \DeclareFixedFont{\Rmm}{T1}{ptm}{m}{n}{2cm}
2 \psboxfill{%
3   \psset{unit=0.1,linewidth=0.2pt}
4   \Kangaroo{PeachPuff}\Kangaroo{PaleGreen}%
5   \Kangaroo{LightBlue}\Kangaroo{LemonChiffon}%
6   \scalebox{-1 1}{%
7     \rput(1.235,4.8){%
8       \Kangaroo{LemonChiffon}%
9       \Kangaroo{LightBlue}%
10      \Kangaroo{PaleGreen}%
11      \Kangaroo{PeachPuff}}}}
12 % A kangaroo of kangaroos...
13 \begin{pspicture}(7.8,2)
14   \pscharpath[linestyle=none,fillloopadd=1,
15     fillstyle=boxfill]
16   {\rput[b](4,0){\Rmm Kangaroo}}
17 \end{pspicture}

```



3.4 Other uses

Other uses can be imagined. For instance, we can use tilings in a sort of degenerate way to draw special lines made by a single or multiple repeating patterns. It might be just a special dashed line, as here with three different dashes:

```

1 \newcommand{\Dashes}{%
2   \psset{dimen=middle}
3   \begin{pspicture}(0,-0.5\pslinewidth)

```

```

4     (1,0.5\pslinewidth)
5     \rput(0,0){\psline(0.4,0)}%
6     \rput(0.5,0){\psline(0.2,0)}%
7     \rput(0.8,0){\psline(0.1,0)}
8   \end{pspicture}}
9
10 \newcommand{\SpecialDashedLine}[3]{%
11   \psboxfill{#3}
12   \Tiling[linestyle=none]
13     {(#1,-0.5\pslinewidth)
14      (#2,0.5\pslinewidth)}}
15 \SpecialDashedLine{0}{7}{\Dashes}
16
17 \psset{unit=0.5,linewidth=1mm,linecolor=red}
18 \SpecialDashedLine{0}{10}{\Dashes}

```

We can also use special patterns in business graphics, as in the following example generated by `PstChart` (Girou, 1993-1998) (see Figure 3).

4 “Dynamic” tiling

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

4.1 Lewthwaite-Pickover-Truchet tiling

We present here as an example the so-called *Truchet* tiling, which is in fact better called *Lewthwaite-Pickover-Truchet (LPT)* tiling, as explained in (Girou, To appear)⁶.

The single prototile is just a square with two opposing circle arcs. This tile obviously has two positions, if we rotate it through 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 its principle, it draws sophisticated curves with strange properties.

Unfortunately, `pst-fill` does not work in a straightforward manner, because the `\psboxfill` macro stores the content of the tile in a `TeX` box, which is static. So the call of the random function is done only once, which explains why only one rotation of the tile is used for all the tiling. Only the one of the two rotations can differ from one drawing to the next ...

```

1 % LPT prototile
2 \newcommand{\ProtoTileLPT}{%

```

⁶ For description of the context, history and references about Sébastien Truchet and this tiling, see (André and Girou, To appear) and specially (André, To appear), (Esperet and Girou, To appear) and (Girou, To appear).

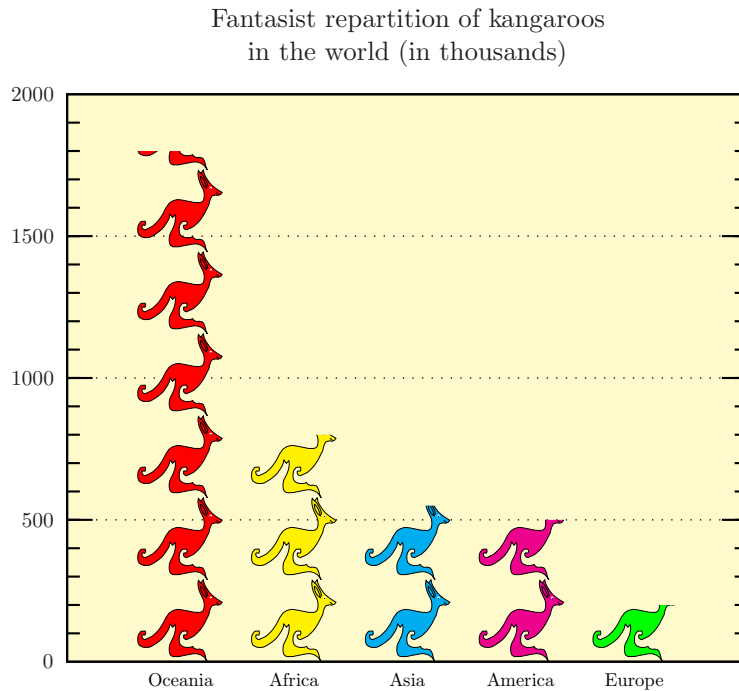
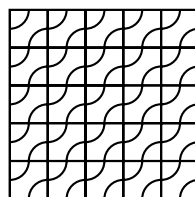


Figure 3: Bar chart generated by PstChart, with bars filled by patterns

```

3  \psset{dimen=middle}
4  \begin{pspicture}(1,1)
5    \psframe(1,1)
6    \psarc(0,0){0.5}{0}{90}
7    \psarc(1,1){0.5}{-180}{-90}
8  \end{pspicture}
9
10 % LPT tile
11 \newcount\Boolean
12 \newcommand{\BasicTileLPT}{%
13   % From random.tex by Donald Arseneau
14   \setranum{\Boolean}{0}{1}%
15   \ifnum\Boolean=0
16     \ProtoTileLPT%
17   \else
18     \rotateleft{\ProtoTileLPT}%
19   \fi}
20
21 \ProtoTileLPT\hfill
22 \rotateleft{\ProtoTileLPT}\hfill
23 \psset{unit=0.5}
24 \psboxfill{\BasicTileLPT}
25 \Tiling{(5,5)}

```



For simple cases, there is a solution to this problem using a mixture of PSTricks and PostScript programming. Here the PSTricks construction `\pscustom{\code{...}}` allows us to insert PostScript code inside the L^AT_EX+PSTricks one. The programming is less straightforward than solving this problem using the basic PSTricks `\multido` macro, but it has the advantage of being noticeably faster, since all tilings operations are done in PostScript, and we are not limited by T_EX memory (the solution without the `pst-fill` package I wrote in 1995 for the colored problem was limited to small sizes for this reason). Note also that `\pslbrace` and `\psrbrace` are PSTricks macros which insert the `{` and `}` characters.

```

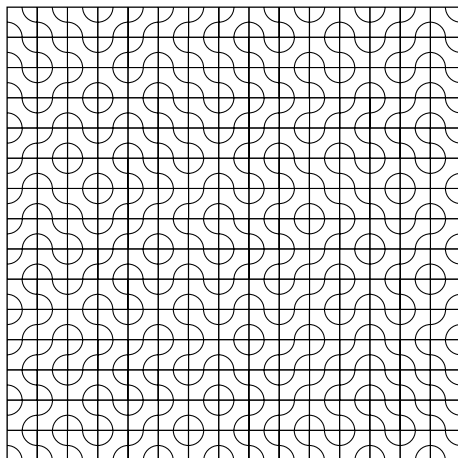
1  % LPT prototile
2  \newcommand{\ProtoTileLPT}{%
3    \psset{dimen=middle}
4    \psframe(1,1)
5    \psarc(0,0){0.5}{0}{90}
6    \psarc(1,1){0.5}{-180}{-90}
7
8  % Counter to change the random seed
9  \newcount\InitCounter
10
11 % LPT tile
12 \newcommand{\BasicTileLPT}{%
13   \InitCounter=\the\time

```

```

14 \pscustom{\code{%
15   rand \the\InitCounter\space
16   sub 2 mod 0 eq \pslbrace}}
17 \begin{pspicture}(1,1)
18   \ProtoTileLPT
19 \end{pspicture}%
20 \pscustom{\code{\psrbrace \pslbrace}}
21 \rotateleft{\ProtoTileLPT}%
22 \pscustom{\code{\psrbrace ifelse}}
23
24 \psset{unit=0.4,linewidth=0.4pt}
25 \psboxfill{\BasicTileLPT}
26 \Tiling{(15,15)}

```



Using the very surprising fact (see (Esperet and Girou, To appear)) that the coloring of these tiles does not depend on their neighbors (even if it is difficult to believe as the opposite seems obvious!) but only on 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 introduced in the `pst-fill` code for *tiling* mode two new accessible PostScript variables, `row` and `column`³, which can be useful in some circumstances, like this one.

```

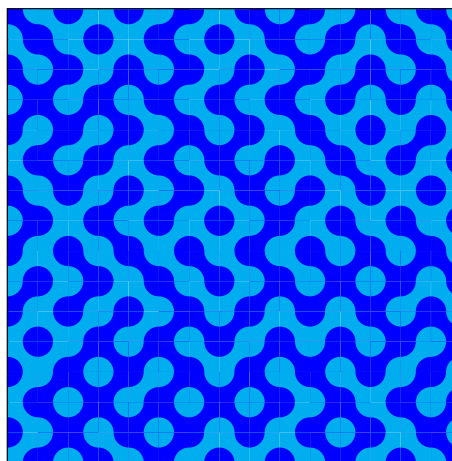
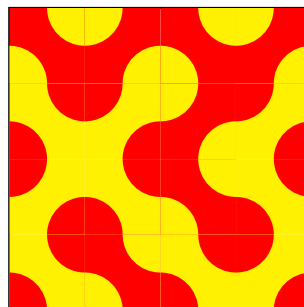
1 % LPT prototile
2 \newcommand{\ProtoTileLPT}[2]{%
3   \psset{dimen=middle,linestyle=none,
4     fillstyle=solid}
5   \psframe[fillcolor=#1](1,1)
6   \psset{fillcolor=#2}
7   \pswedge(0,0){0.5}{0}{90}
8   \pswedge(1,1){0.5}{-180}{-90}}
9
10 % Counter to change the random seed
11 \newcount\InitCounter
12
13 % LPT tile
14 \newcommand{\BasicTileLPT}[2]{%
15   \InitCounter=\the\time
16   \pscustom{\code{%
17     rand \the\InitCounter\space sub 2

```

```

18     mod 0 eq \pslbrace
19     \row \column add 2 mod 0 eq \pslbrace}}
20 \begin{pspicture}(1,1)
21   \ProtoTileLPT{#1}{#2}
22 \end{pspicture}%
23 \pscustom{\code{\psrbrace \pslbrace}}
24 \ProtoTileLPT{#2}{#1}%
25 \pscustom{\code{%
26   \psrbrace ifelse \psrbrace \pslbrace
27   \row \column add 2 mod 0 eq \pslbrace}}
28 \rotateleft{\ProtoTileLPT{#2}{#1}%
29 \pscustom{\code{\psrbrace \pslbrace}}
30 \rotateleft{\ProtoTileLPT{#1}{#2}}%
31 \pscustom{\code{\psrbrace ifelse
32   \psrbrace ifelse}}
33
34 \psboxfill{\BasicTileLPT{red}{yellow}}
35 \Tiling{(4,4)}
36
37 \vspace{2mm}
38 \psset{unit=0.4}
39 \psboxfill{\BasicTileLPT{blue}{cyan}}
40 \Tiling{(15,15)}

```



Another classic example is generation of coordinates and labelling for a grid. Of course, it is possible to do it directly in PSTricks using nested `\multido` commands, and it would clearly be easy to program. Nevertheless, for users who have a little knowledge of PostScript programming, this method offers an alternative which is useful for large cases,

because it will be noticeably faster and use less computer resources.

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

```

1 % \Escape will be the \ character
2 {\catcode'\!=0\catcode'\=\!1!gdef!Escape{\}}
3
4 \newcommand{\ProtoTile}{%
5   \Square%
6   \pscustom{%
7     \moveto(-0.9,0.75) % In PSTricks units
8     \code{%
9       /Times-Italic findfont 8 scalefont setfont
10      (\Escape) show row 3
11      string cvs show (, ) show column 3 string
12      cvs show (\Escape) show}
13     \moveto(-0.5,0.25) % In PSTricks units
14     \code{%
15       /Times-Bold findfont 18 scalefont setfont
16       1 0 0 setrgbcolor % Red color
17       /center {dup stringwidth pop 2
18         div neg 0 rmoveto} def
19       row 1 sub x2 mul
20       column add 3 string cvs center show}}
21 \psboxfill{\ProtoTile}
22 \Tiling{(6,4)}

```

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

```

1 \newcommand{\Pattern}[1]{%
2   \begin{pspicture}(-0.25,-0.25)(0.25,0.25)
3     \rput{*0}{\psdot[dotstyle=#1]}
4     \end{pspicture}}
5 \newcommand{\West}{\Pattern{o}}
6 \newcommand{\South}{\Pattern{x}}
7 \newcommand{\Central}{\Pattern{+}}
8 \newcommand{\North}{\Pattern{square}}
9 \newcommand{\East}{\Pattern{triangle}}
10
11 \newcommand{\Cross}{%
12   \pspolygon[unit=0.5,linewidth=0.2,
13     linecolor=red]
14     (0,0)(0,1)(1,1)(1,2)(2,2)(2,1)(3,1)(3,0)
15     (2,0)(2,-1)(1,-1)(1,0)}
16
17 \newcommand{\StylePosition}[1]{%
18   \LARGE\textcolor{red}{\textbf{#1}}}
19
20 \newcommand{\SubDomain}[4]{%

```

```

21 \psboxfill{#4}
22 \begin{psclip}{\psframe[linestyle=none]#1}
23   \psframe[linestyle=#3](5,5)
24   \psframe[fillstyle=boxfill]#2
25 \end{psclip}}
26
27 \newcommand{\SendArea}[1]{%
28   \psframe[fillstyle=solid,fillcolor=cyan]#1}
29
30 \newcommand{\ReceiveData}[2]{%
31   \psboxfill{#2}
32   \psframe[fillstyle=solid,fillcolor=yellow,
33     addfillstyle=boxfill]#1}
34
35 \newcommand{\Neighbor}[2]{%
36   \begin{pspicture}(5,5)
37     \rput{*0}(2.5,2.5){\StylePosition{#1}}
38     \ReceiveData{(0.5,0)(4.5,0.5)}{\Central}
39     \SendArea{(0.5,0.5)(4.5,1)}
40     \SubDomain{(5,2)}{(0.5,0.5)(4.5,3)}
41       {dashed}{#2}%
42     % Receive and send arrows
43     \pcarc[arcangle=45,arrows=->]
44       (0.5,-1.25)(0.5,0.25)
45     \pcarc[arcangle=45,arrows=->,
46       linestyle=dotted,dotsep=2pt]
47       (4.5,0.75)(4.5,-0.75)
48   \end{pspicture}}
49
50 \psset{dimen=middle,dotscale=2,fillloopadd=2}
51 \begin{pspicture}(-5.7,-5.7)(5.7,5.7)
52   % Central domain
53   \rput(0,0){%
54     \begin{pspicture}(5,5)
55       % Receive from West, East, North and S.
56       \ReceiveData{(0,0.5)(0.5,4.5)}{\West}
57       \ReceiveData{(4.5,0.5)(5,4.5)}{\East}
58       \ReceiveData{(0.5,4.5)(4.5,5)}{\North}
59       \ReceiveData{(0.5,0)(4.5,0.5)}{\South}
60       % Send area for West, East, North and S.
61       \SendArea{(0.5,0.5)(1,4.5)}
62       \SendArea{(4,0.5)(4.5,4.5)}
63       \SendArea{(0.5,0.5)(4.5,1)}
64       \SendArea{(0.5,4)(4.5,4.5)}
65       % Central domain
66       \SubDomain{(5,5)}{(0.5,0.5)(4.5,4.5)}
67         {solid}{\Central}
68       % Redraw overlapped lines
69       \psline(1,0.5)(1,4.5)
70       \psline(4,0.5)(4,4.5)
71       % Two crosses
72       \rput(1.5,4){\Cross}
73       \rput(2,2){\Cross}
74     \end{pspicture}}
75   % The four neighbors
76   \rput(0,5.5){\Neighbor{N}{\North}}
77   \rput{-90}(5.5,0){\Neighbor{E}{\East}}
78   \rput{90}(-5.5,0){\Neighbor{W}{\West}}
79   \rput{180}(0,-5.5){\Neighbor{S}{\South}}
80 \end{pspicture}

```

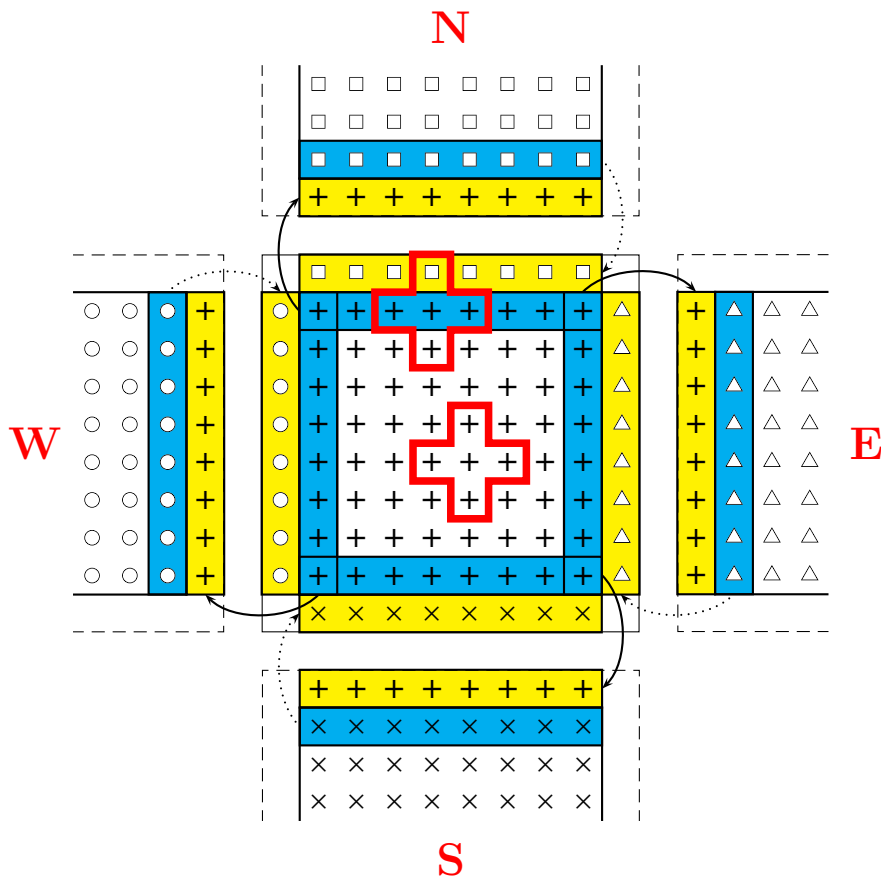


Figure 4: Communication scheme to solve the Poisson equation on a distributed memory computer

4.2 A complete example: the Poisson equation

To finish, we show in Figure 4 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. The code (listed below) also shows some useful and powerful techniques for PSTricks programming (look especially at the way some higher level macros are defined, and how the same object is used to draw the four neighbors).

References

- Adobe, Systems Incorporated. *PostScript Language Reference Manual*. Addison-Wesley, 2 edition, 1995.
- 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, André. *Le monde des pavages*. ACL Éditions, 1997.
- Girou, Denis. “PstChart. Business charts in (L)T_EX + PostScript with PSTricks”. <http://www.tug.org/applications/PSTricks/PstChart>, 1993–1998.
- Girou, Denis. “Présentation de PSTricks”. *Cahiers GUTenberg* **16**, 21–70, 1994.
- Goossens, Michel, S. Rahtz, and F. Mittelbach. *The L^AT_EX Graphics Companion*. Addison-Wesley, 1997.
- Grünbaum, Branko and G. Shephard. *Tilings and Patterns*. Freeman and Company, 1987.
- Hoenig, Alan. *T_EX Unbound: L^AT_EX & T_EX Strategies, Fonts, Graphics, and More*. Oxford University Press, 1998.
- Rose, Kristoffer H. and R. Moore. “Xy-pic. Pattern and Tile extension”. Available from CTAN, [macros/generic/diagrams/xy-pic](http://www.ctan.org/macros/generic/diagrams/xy-pic), 1991–1998.

- van der Laan, Kees. “Paradigms: Just a little bit of PostScript”. *MAPS* **17**, 137–150, 1996.
- van Zandt, Timothy. “PSTricks. PostScript macros for Generic TeX”. Available from CTAN, `graphics/pstricks`, 1993.
- van Zandt, Timothy and D. Girou. “Inside PSTricks”. *TUGboat* **15**(3), 239–246, 1994.
- Wang, Hao. “Games, Logic and Computers”. *Scientific American* pages 98–106, 1965.

◇ Denis Girou
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`