PMGRAPH.STY: some useful macros which extends the LaTeX environment `picture`

Version 1.0

A.S.Berdnikov
berd@ianin.spb.su

O.A.Grineva
olga@ianin.spb.su

The original TeX/LaTeX possibilities to create pictures are relatively poor, and there are many extensions (`epic/eepic`, `pictex`, `drawtex`, `xypic`, `mfpic`, etc.) which were created to extend its possibilities to a higher level. The macro PMGRAPH.STY (*poor-man-graphics*) which are described here are not so general as the ones cited above. They manupulate with the pseudo-graphical fonts which are used by generic LaTeX without additional extensions — mainly because the variations of PiCTeX, METAFONT and new graphical font themes are already realized by other authors and on sufficiently higher level. To some extend the purpose of our work was to see how far it is possible to move in the development of new useful graphical primitives for LaTeX *without* the investment of the external graphical tools.

The style file PMGRAPH.STY includes the following features:

- the vectors with a set of slopes which is as general as the line slopes implemented in LaTeX;
- the vectors with an arrow at the beginning, at the middle or at the end of vector with various orientations of the arrow;
- the circles and circular arcs with nearly arbitrary diameter using magnified `circle` and `circlew` LaTeX fonts;
- the 1/4 circular arcs correctly positioned at the centrum or at the corner;
- extended set of frames which include various corner style and the optional multiple frame shadows with a variety of styles;
- tools which enable the user to extend the variety of frame styles and the shadow styles as far as his/her fantasy allows it;
- automatic calculation of the picture size in terms of the current text width — including the `picture` inserted inside list environments.

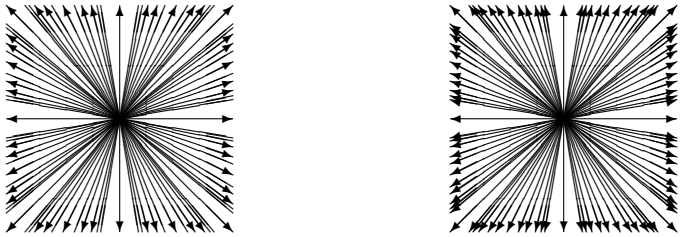

Figure 1: LaTeX and PMGRAPH vectors

$(1,1)$	$(1,1)$	$(4,1)$	$(4,1)$	$(5,3)$	$(3,2)$
$(2,1)$	$(2,1)$	$(4,3)$	$(4,3)$	$(5,4)$	$(4,3)$
$(3,1)$	$(3,1)$	$(5,1)$	$(4,1)$	$(6,1)$	$(4,1)$
$(3,2)$	$(3,2)$	$(5,2)$	$(3,1)$	$(6,5)$	$(4,3)$

Table 1: Relation between line slopes and approximate vector slopes

Even not very complicated, these macros appears to be useful in our work, and it seems that they can be useful for other TeX-users too.

Vectors

The number of angles for inclined lines which can be used in LaTeX is limited to great extend, but the number of angles for *vectors* is limited even more. The variety of vectors can be extended if instead of the *strictly* inclined arrows at the end of the inclined line the arrow with the *approximate* inclination is added. Corresponding changes are incorporated in PMGRAPH where the relation between strict inclinations and approximate inclinations are shown in Table 1. The corrections require the modifications of the internal LaTeX commands `\@svector`, `\@getlarrow`, `\@getrarrow` and the command `\vector` itself. As a result the command `\vector` starts to draw the vectors for all inclinations valid for LaTeX lines as it is shown on Fig. 1. The vectors are not so ideal as it is required by TeX standards, but the results are acceptable for all inclinations except $(6,1)$.

LaTeX allows to put an arrow just at the end of the vector. The command `\Vector` enables to put along the vector *arbitrary* arrows with different orientation (see Fig. 2). The predefined arrow styles assign a letter to each position and orientation of the arrow along the `Vector`.

The arrows shown on Fig. 2 are drawn by the commands

```
\begin{picture}(300,40)
\put(20,5){\Vector[bme](1,0){100}}
\put(20,30){\Vector[BME](1,0){100}}
```

Figure 2: Multi-arrow vectors

```
\put(170,5){\Vector[xmMZ](1,0){100}}
\put(170,30){\Vector[XmMz](1,0){100}}
. . . . . .
```

Letter `e` corresponds to normally oriented arrow at the end of vector, `E` — to reverse oriented arrow, `b` and `B` — to (normally and reverse oriented) arrows at the beginning of the vector, `m` and `M` — to the arrows at the middle, etc. The list of letters as the optional parameter produces the set of arrows along the `Vector`. It is possible to create user-defined styles of arrows using the commands `\VectorStyle` and `\VectorShiftStyle` (where the parameters in square brackets are *obligatory*, not *optional*):

`\VectorStyle[`*style-char*`]{`*shift-char*`}{`*position*`}{`*orientation*`}`

- *style-char* is the character which is assigned to vector style;
- *shift-char* is the character which defines the relative shift of the arrow with respect to *position* — see command `\VectorShiftStyle` below;
- *position* is the real value which defines the relative position of the arrow along the vector (0.0 means starting point of the vector, 1.0 means end point of the vector) which usually is in a range 0..1 but can be greater 1 or less 0 as well;
- *orientation* is the character which defines the orientation of the arrow with respect to the standard direction of the vecrtor: `b` means *backward* direction, `f` (or any other character) means forward direction.

`\VectorShiftStyle[`*style-char*`]{`*shift*`}`

- *style-char* is the character which is assigned to vector-shift-style;
- *shift* is the relative shift in `pt` of the arrow along the arrow direction with respect to the positioning point (it is necessary to note that the length of the arrow body in LaTeX is equal to `4pt`).

Examples:

- standard *vector-shift-styles*:

  `\VectorShiftStyle[e]{0pt}` — style '`e`' means that the end of the arrow is positioned strictly at the point, specified by the parameter *position*;

`\VectorShiftStyle[b]{4pt}` — style '`b`' means that the backside of the arrow is positioned at the point, specified by the parameter *position*;

`\VectorShiftStyle[m]{3pt}` — style '`m`' means that the middle of the arrow body is positioned at the point, specified by the parameter *position*;

`\VectorShiftStyle[E]{-2pt}` — style '`E`' means that the end of the arrow is positioned a little bit before (i.e., by `2pt`) the point, specified by the parameter *position*;

`\VectorShiftStyle[B]{6pt}` — style '`B`' means that the backside of the arrow is positioned a little bit after (i.e., by `2pt`) the point, specified by the parameter *position*.

- standard *vector-styles*:

  `\VectorStyle[e]{e}{1.0}{f}` — style '`e`' means that the end of the arrow is positioned at the end of the vector, and its orientation is along the vector orientation;

  `\VectorStyle[E]{b}{1.0}{b}` — style '`E`' means that the backside of the arrow is positioned at the end of the vector, and its orientation is rotated by 180° with respect to the vector orientation;

  `\VectorStyle[b]{b}{0.0}{f}` — style '`b`' means that the backside of the arrow is positioned at the beginning of the vector, and its orientation is along the vector orientation;

  `\VectorStyle[B]{e}{0.0}{b}` — style '`B`' means that the end of the arrow is positioned at the beginning of the vector, and its orientation is rotated by 180° with respect to the vector orientation;

  `\VectorStyle[m]{m}{0.0}{f}` — style '`m`' means that the middle of the arrow body is positioned at the middle of the vector, and its orientation is along the vector orientation;

  `\VectorStyle[M]{m}{0.0}{b}` — style '`M`' means that the middle of the arrow body is positioned at the middle of the vector, and its orientation is rotated by 180° with respect to the vector orientation;

  `\VectorStyle[x]{E}{1.0}{f}` — style '`x`' means that the end of the arrow is positioned a little bit before the end of the vector, and its orientation is along the vector orientation;

  `\VectorStyle[X]{B}{1.0}{b}` — style '`X`' means that the backside of the arrow is positioned a little bit before the end of the vector, and its orientation is rotated by 180° with respect to the vector orientation;

  `\VectorStyle[z]{B}{0.0}{f}` — style '`z`' means that the backside of the arrow is positioned a little bit after the beginning of the vector, and its orientation is along the vector orientation;

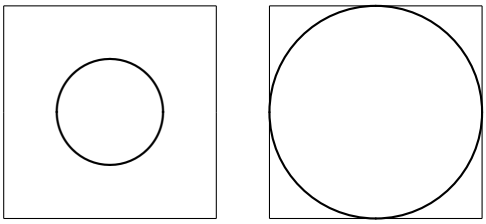

Figure 3: Magnified circles

`\VectorStyle[Z]{E}{0.0}{b}` — style '`Z`' means that the end of the arrow body is positioned a little bit after the beginning of the vector, and its orientation is rotated by 180° with respect to the vector orientation.

Circles

The range of the diameters for circles and disks (black circular blobs) available in LaTeX is very restricted. It can be enlarged by using the magnified pseudo-graphical LaTeX fonts if the User does not have something better at his/her disposal like `curves.sty`, PiCTeX or MFPiC. The disadvantage of this method is that the width of the lines is magnified too which is inconsistent with the rigorous TeX accuracy requirements, but for *poor man graphics* these circles can be satisfactory.

The scaling of circular fonts is performed by the commands

`\scaledcircle{`*factor*`}`
`\magcircle{`*magstep*`}`

which are indentical to TeX commands

`\font ... scaled` *factor*
`\font ... scaled \magstep` *magstep*

The valid *magstep* values are `0`, `h`, `1`, `2`, `3`, `4`, `5`. The values *factor*=1000 and *magstep*=0 correspond to *one-to-one* magnification. The circle magnification like other TeX commands returns to its previous value outside the group inside which it was changed.

To calculate properly the circle character code after the magnification it was necessary to redefine some internal LaTeX commands like `\@getcirc` and `\@circ`. To reflect in magnified fonts the changes of the line thickness, the commands `\thinlines` and `\thicklines` are corrected also.

The example on Fig. 3 is produced by

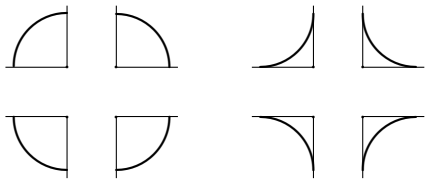

Figure 4: 90° circular segments

```
\setlength{\unitlength}{1pt}
\begin{picture}(200,100)(-100,-50)
\put(-50,0){\thicklines\circle{80}}
\put(-50,0){\squareframe{40}}
\magcircle{4}
\put(50,0){\thinlines\circle{80}}
\put(50,0){\squareframe{40}}
\end{picture}
```

where `\squareframe` is the user-defined command which draws the square with the specified side and the centrum at (0,0). It shows how the usage of the magnified circles enables to overcome the upper limit 40pt of the diameter of the LaTeX cirles. It is necessary to note that the thickness of the `\thinline` circles after magnification with `\magcircle{4}` corresponds approximately to the thickness of the ordinary `\thickline` circles (`\magstep4` ≈ 2000).

Additional macro enable to draw 90° quaters of the circles explicitly without tricky refinement of the parameters of the command `\oval`:

`\trcircle{`*diam*`}` $\longrightarrow$ `\oval[tr]...`
`\brcircle{`*diam*`}` $\longrightarrow$ `\oval[br]...`
`\tlcircle{`*diam*`}` $\longrightarrow$ `\oval[tl]...`
`\blcircle{`*diam*`}` $\longrightarrow$ `\oval[bl]...`

The centrum of the circular arc is positioned strictly at the point which is the argument of the corresponding `\put`. The commands `\TRcircle`, `\BRcircle`, `\TLcircle`, `\BLcircle` draw the 90° circular quaters with the reference point positioned at the corner instead of the centrum. Similarly, the commands

`\tlsector`, `\TLsector`, `\blsector`, `\BLsector`, ...

draw circular segments together with horizontal and vertical radii. The proper positioning of the circular segments requires special precausions since it is necessary to take into account the line thickness and the specific alignment of the circular elements inside the character boxes.

The example on Fig 4 shows the usage of these commands:

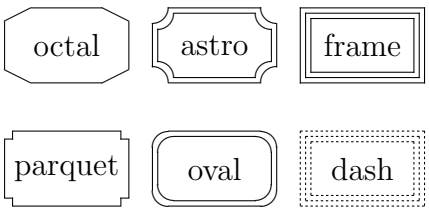

Figure 5: Examples of frame styles

```
\begin{picture}(200,60)(-100,-30)
\put(-60,10){\thicklines\tlcircle{50}}
\put(-60,10){\circle*{1}}
\put(-60,10){\line(-1,0){25}}
\put(-60,10){\line(0,1){25}}
\put(40,10){\thicklines\BRcircle{50}}
\put(40,10){\circle*{1}}
\put(40,10){\line(-1,0){25}}
\put(40,10){\line(0,1){25}}
... ... ...
```

The actual diameter of the circular segment is adjusted like it is done with the circles. The commands `\scaledcircle` and `\magcircle` affect the thickness and the diameter of these circular segments also.

Frames

The set of frames which are available in LaTeX is enhanced in PMGRAPH — except solid and dashed rectangular frames it is possible to draw double and tripple frames in a variety of styles (Fig. 5). The commands `\frameBox`, `\ovalBox`, `\octalBox`, `\astroBox`, `\parquetBox` have the same structure as the command `\framebox`, but they draw the corresponding fancy frames:

```
\put(0,0){\ovalBox(100,50){oval}}
\put(70,0){\astroBox(100,50){astro}}
. . . . . .
```

The ordinary solid frame is drawn by `\frameBox`, the double and triple frames are drawn by `\frameBoX` and `\frameBOX`, respectively. Similar commands exist for double and triple fancy frames. The User can prepare the personal macro commands to draw frame corners and extend the variety of fancy frames up to the limit of his/her fantasy.

More exotic variant of a frame can be created using the commands `\rombBox`, `\rombBoX` or `\rombBOX` as it is shown on Fig. 6. The style (i.e., inclination of the romb sides) and the distance between multiple rombs are set by the command `\rombboxstyle`

Figure 6: Romb-style frames

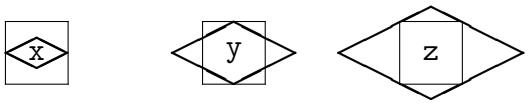

Figure 7: Alignment of romb boxes

`\rombboxstyle(`Δx`,`Δy`,`*len*`)` — defines the inclination for the romb boxes and for the corners of the tt octal frames and shadows. Parameters Δx, Δy specifies the inclination, and the parameter *len* — the length of the inclined corners (for `octal` frames and shadows only) in a style similar to the command `\line(`Δx`,`Δy`)`{*len*}.

with the default settings as

```
\rombboxstyle(2,1,2pt)
```

The alignment of the romb around the box specified for these commands can be varied using additional optional parameter(see Fig. 7. The full format of the `rombbox` commands is:

`\rombBox[`*char*`]`$(\Delta X, \Delta Y)$`{`*text*`}`

where *char* -is one-character parameter which defines the alignment of the romb frame with respect to rectangle $(\Delta X, \Delta Y)$: `x` (default value) means that the x-axis coinsides with the x-axis of the rectangle, `y` means that the y-axis coinsides with the y-axis of the rectangle, `z` means that the corners of the rectangle are at the sides of the romb frame.

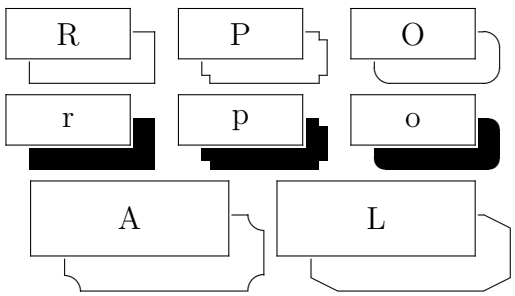

Figure 8: Examples of shadows

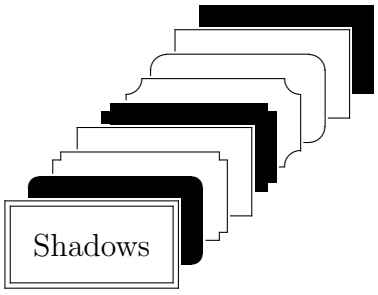

Figure 9: Multiple shadows

Each rectangular box has the optional parameter which enable to specify the "shadows" around this box. Each shadow style has a special letter, and the list of letters as the optional parameter draws a list of shadows. The standard shodow types are shown on Fig. 8. It is possible to draw several shadows of different types and around arbitrary corner of the frame as it is shown on Fig. 9:

```
\unitlength=10pt
\begin{picture}(20,15)
\shadowcorner{B}
\put(0,0){\frameBoX[oPR...](10,5){...}}
\end{picture}
```

The parameter of the shadows — thickness, corner size, additional shift, etc., — can be varied by the following User commands:

`\framesep`{*dist*} — set the distance between double and triple frames. It can be negative as well as positive. Default value: `2pt`}.

`\shadowsep`{*dist*} — set the gap distance between the frame and the shadow or between multiple shadows. Default value: `1pt`}.

`\shadowsize`{*dist*} — set the depth of the shadow. Default value: `5pt`}.

`\shadowshrink`{*factor*} — set the contraction factor for the subsequent shadows. Default value: `1`}.

`\shadowcorner`{*char*} — set the corner for the shadows. Valid values: `A`, `B`, `C`, `D`. Default value: `\shadowcorner{A}`.

`\RoundCorner`{*radius*} — set the *radius* for the circular arcs at the corners of oval frames and shadows with rounded corners. Default value: `6pt{5pt}`.

`\DiskCorner`{*diam*} — set the *diameter* for the bulbs at the corners of black shadows with rounded corners. Default value: `8pt{5pt}`.

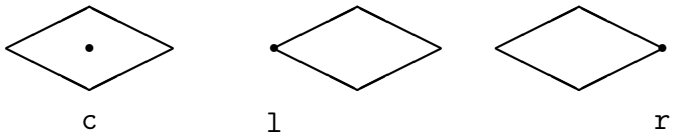

Figure 10: Alignment of rombes

`\LineCorner{`*len*`}` — set the *length* for the inclined corners of octal frames and shadows. Default value: `10pt{10pt}`.

`\RectCorner{`*size*`}` — set the *size* for the parquette corners of octal frames and shadows. Default value: `3pt{5pt}`.

Rombs

Special command enable to draw rombs (see Fig. 10):

`\put(`x`)(`y`) {\romb[`*pos*`](`Δx`,`Δy`){`*len*`}}`

where:

(x, y) — position of the romb inside `picture`;
pos — one-character option which shows the alignment of romb with respect to (x, y): `r` means right corner, `l` means left corner; `c` means center (default);
$(\Delta x, \Delta y)$ and *len* are the parameters which define the inclination and the length of the romb side (similarly to `\line`).

Automatically scaled pictures

The idea of the macros which are responsible for these functions is to calculate the `\unitlength` value in terms of the *relative fraction* of the page width instead of explicit specifying its value in points, centimeters, inches, etc.

The command `\pictureunit[`*percent*`]{`*x-size*`}` selects the value of the variable `\unitlength` so that the picture which is *x-size* units in width occupies *percent* width of the paper. The environment `Picture` combines the automatic calculation of the `\unitlength` with the `\begin{picture}` – `\end{picture}`. By default *percent*=100 is used which corresponds to 90% of the paper width. The default *percent* value can be redefined by the command

`\def\defaultpercent{`*percent*`}`.

Examples:

```
\pictureunit[75]{120}
\begin{picture}(120,80)
...
\end{picture}

\begin{Picture}[75](120,80)
...
\end{Picture}
```

These macros are inspired by `fullpict.sty` by Bruce Shawyer. Carefull examination of the file `fullpict.sty` shows some bugs/warnings which require correction:

- each automatic scaling of `\unitlength` allocates a new counter;
- automatic scaling uses `\textwidth` as the reference width which results to improper functioning inside list and `minipage` environments;
- the environments `fullpicture`, `halfpicture` and `scalepicture` are centered internally with `\begin{center}` — `\end{center}` which prevents the proper positioning of the picture in most cases.

The PMGRAPH.STY macros calculates the dimension `\unitlength` using the value `\hsize`, and as a result it works corectly also for `twocolumn` mode, inside the *list* environments `itemize`, `enumerate`, etc. (for example, all the figures in this paper are drawn using the environment `Picture`). The automatic centering and repeatedly allocation of the registers are corrected as well.

Acknowledgements

The authors are grateful to Dr. Kees van der Laan for the possibility to present the results of our research at the EUROTEX'95 (Aarnhem, Netherland).

One of the authors (A.S.Berdnikov) would like to express his warmest thanks to Dr. A.Compagner from the Delft University of Technology who spent a lot of his time and efforts trying to transform two naive students from Russia (namely, him and his co-worker Sergey Turtia) into serious scientists.

This research was partially supported by a grant from the Dutch Organization for Scientific Research (NWO grant No 07–30–007).