PMPIC — *Poor Man' Picture* package, the extension of the standard LaTeX picture environment

Olga A. Grineva
Institute of Analytical Instrumentation,
St. Petersburg, Russia
olga@p50.f924.n5030.z2.fidonet.org

Galina V. Mitina
All-Russian Institute of Plant Protection,
St. Petersburg, Russia
Galina@p10.f108.n5030.z2.fidonet.org

Alexey V. Filippov
St. Petersburg State Institute of Fine Mechanics and Optics,
St. Petersburg, Russia
alf@ptslab@ioffe.rssi.ru

Alexander S. Berdnikov
Institute of Analytical Instrumentation,
St. Petersburg, Russia
berd@ianin.spb.su

Abstract

While there are many powerful tools to draw graphics in LaTeX, most ordinary users still prefer to use the native LaTeX' environment `picture` as the main tool to make pictures. To help the users who still use standard LaTeX drawing tools, the package PMPIC was created. Its major extension is the possibility to draw the lines with arbitrary slopes without visually distinguishable wire-edges (this requires the 8-bit line fonts instead of 7-bit LaTeXnative fonts LINE10 and LINEW10), and the range of available circular elements is extended in a similar way. In addition the User can find in PMPIC such attractive features as '*turtle graphics*', arithmetics with variables (numbers, points, vectors, etc.), a variety of line and vector styles, fancy frames with shadows (the latter is under development now).

Introduction

Graphics is the inevitable component of the document, and LaTeX offers many powerful tools to include (scientific) graphics in your documents (the packages PICTEX [1], DRATEX [2], XYPIC [3], MFPIC [4], PSTRICS [5], etc.). But as it follows from the documents available in Internet, most ordinary users still prefer to use the native LaTeX' environment `picture` with some useful extensions (for example, EPIC [6], EEPIC [7], CURVES [8], etc.).

The most annoying features of LaTeX pictures are the limitation for the range of slopes available for lines and vectors, unpredictable disappearing of extra short lines, quite limited range of circular elements. Some time ago we tried to overcome these limitations (package PMGRAPH [9]) using the magnified LaTeX graphical fonts. Although that solution was satisfactory for high resolution output with medium requirements to picture quality, it appears to be quite bad when the .pdf-file from .tex-file is created — all the "wire-edges" are clearly visible on screen at sufficient magnification. (And PMGRAPH also has such undesirable side effect that the *thickness* of lines and circles is magnified as well.)

Similar problems arise in .pdf-files with EPIC and EEPICEMU — the overlappings of individual line segments are clearly visible. It is strange, but the pictures created from tiny points by PICTEX and DRAWTEX are also not too good in .pdf-files but for another reason — while it takes a lot of time and resources for TeX compiler to create the .dvi-file, it takes no less time and resources from .pdf-viewer to display it on screen. To analyze from this point of view the full set of drawing facilities avail-

Polska Grupa Użytkowników Systemu TeX, 2000 (http://www.GUST.org.pl)

able in LaTeX, it is necesary to note that conversion of MFPIC output into input data acceptable for .pdf-document requires expert level skills, and the same is true for PSTRICKS (if it is possible at all). But, for example, XYPIC works without problems, and the same is true for METAPOST and interactive drawing programs which produce the .ps and .eps output (lately converted to .pdf by GHOSTSCRIPT, for example).

To help the users who still use standard LaTeX drawing tools (i.e., the `picture` environment with its minimal extensions), the style file PMGRAPH was modified and extended. The result of this modification is the LaTeX 2$_\varepsilon$ package PMPIC described here. Its major extension is the possibility to draw the lines with arbitrary slopes without visually distinguishable wire-edges — it requires the special 8-bit line fonts instead of 7-bit LINE10 and LINEW10 used currently by LaTeX. Similarly the range of available circles and disks is highly extended in PMPIC by using 8-bit fonts. While most of other extensions (like frames with multiple shadows) are just copied from PMGRAPH, the User can find here some new and attractive features like '*turtle graphics*' and arithmetics with variables (numbers, points, vectors, directions, etc.) as well.

Lines and vectors from mosaic elements

It is more or less clear that format of .dvi-files does not allow to code flexible graphics. You can use extensions specific for your .dvi-driver (POSTSCRIPT, TPIC and EMTEX `\special`'s), you can compose the picture from individual tiny points (PICTEX, DRATEX, CURVES), you can assemble your picture from pseudo-graphical font elements of a finite size (environment `picture`, package XYPIC). Finally, you can generate your picture 'as if' it is a block of text composed from special letters (BM2FONT, MFPLOT) or a single letter (MFPIC). And you also can use external programs which perform this work for you automatically, not manually.

Although the usage of a fixed and finite set of graphical elements with finite size seens to be less flexible, we decided to use just this approach. It is compatible with the original LaTeX approach, it enables to keep the amount of the elements used for a single picture at a reasonable level, it does not require .dvi-extensions and external utilities, it is absolutely transparent and movable between platforms.

But just the same way leads LaTeX to its restricted set of slopes and lengths for lines and vectors, restricted range of available circles, etc., isn't it? Yes, you are right. But it is true as soon as you do not take the *finite* thickness of your graphical elements. For example the package EPIC [6] demonstrates that is it *possible* to make the lines with infinite set of slopes using the LaTeX graphical fonts, and only the highly restricted subset of available graphical primitives makes these lines to be with ugly edges. As soon as the range of available slopes is big enough, there are no visually detected edges on your picture!

Let us consider this statement in more details. Supose there is the line segment of length L and thickness W — for simplicity we can assume it to be the rectangular block without round edges (see Fig. 1). Suppose its slope is different by the angle $\Delta\alpha$ from the ideal slope we would like approximate with this line segments. Then there is the finite shift of the end of the previous line segment and the start of the next line segment which is calculated as $\Delta W = L \cdot \tan(\Delta\alpha) \approx L \cdot \Delta\alpha$ (where $\Delta\alpha$ is in radians). If there is the finite set of slopes produced as the uniform division of the angle $\pi/2$ by N, in the worst case $\Delta\alpha \leq \pi/(4N)$, and for $L = 8pt$, $W = 0.4pt$, $N = 64$ we get $\Delta W < 0.1pt$ — i.e., less than $1/4 W$.

These parameters were selected for 8-bit line font used by PMPIC. Since the slopes in a range $-\pi/2 \ldots \pi/2$ copies the slopes in a range $\pi/2 \ldots 3\pi/2$, we can add short segments with the length $2pt$ in addition to ordinary segments with the length $8pt$ (short line segments are used when the User requires the line which is shorter than $8pt$). Moreover, since there is no necessity to make the lines with the slopes of $0°$ and $\pm 90°$ (they are substituted by TeX rules), we have some free cells in our 8-bit font for special symbols — for example, for a single dot with diameter equal to line thickness which enables to construct the line with the length less than $2pt$ from individual points if the User really needs it.

Since the slopes occupy all 256 cells of line font, the arrows (vector headings) are moved to a separate font. In this case we need to fill the whole $360°$ circle by arrow directions since the arrows are not symmetrical, and we also need the arrow headings for $0°$, $\pm 90°$ and $180°$ directions. But simultaneously we can allow greater error in arrow tip position since it shows the final point and is not used for fine connection with other elements. (Moreover, we can make the arrow end a little bit round to mask the shift of its tip from the strict end point. And the arrows are shorter than line segments which also makes the error a little bit smaller.)

As a result the angle of $\pi/2$ is divided by 16 to make the discrete set of arrow slopes (it gives the error of the final point $\approx W$), but the vector font contains arrows for *four* various arrow styles:

Polska Grupa Użytkowników Systemu TeX, 2000 (http://www.GUST.org.pl)

small and big, filled and outlined. Moreover, there are *two* different arrow fonts used by PMPIC: one follows the style suggested by David Solomon [10], and one follows the style designed for LATEX by Leslie Lamport. (See section *More thickness, more circles, more fonts* for more details.)

Circles and disks from mosaic elements

In a similar way the 7-bit LATEX circle font is substituted by extended 8-bit font with more circle sizes (the greatest available radius is 98*pt*). Moreover, the disks (i.e., circles filled by black, or bullets) are moved to a separate font – otherwise they will be copied several times when we create several circle fonts for various line thickness (see section *More thickness, more circles, more fonts*).

Except the size range, the *variability* of the circular elements is also extended. For example, instead of $1/4$ circular arcs there are $1/8$ circular arcs for the circles with radii $2pt \div 34pt$ (they are used for rounded corners for *turtle graphics* — see section *User-friendly syntax and turtle graphics*), for radii $38pt \div 98pt$ there are $1/4$ circular arcs, and only for extra small radii there are full circles in circle font.

The bullet font covers just the same range of radii, but its encoding is different: it contains only $1/4$ circular segments, but there are astro-shaped figures as well as circular-shaped ones (astro-shaped corners are used for shadows and fancy frames — see section *Boxes, frames, shadows*). Moreover, for some radii there are $1/4$ black rings instead of $1/4$ circular segments — so, for big radii the bullet is composed from sequential layers of final thickness, but this approach enables to make black rings for our pictures when we need them.

While the circular elements vary in their size so strongly, some problems appear when we make the font from such elements. The first one deals with the fact that .tfm-file allows only 16 heights and 16 depths to be encoded. This problem is solved by assigning the zero height and zero depth for each element — but it produces problems when we try, for example, print the whole font by *Plain TEX* and `testfont` utility.

The second problem relates to the fact that the METAFONT characters cannot be infinitely big when expressed in pixels — whey can occupy ±32767 pixels in X- and Y-direction, no more. As a result the extremely big 'characters' for the devices with extremely high resolution fail (or you should use the POSTSCRIPT technology, as it is done usually in real life, for such devices). This is the reason to restrict the range of radii by $\approx 100pt$, but we can do more: in addition the $1/4$ circular elements are shifted for big radii in such a way, that they are centered with respect its actual height and width. (Namely, the $1/4$ segment with radius 98*pt* is $\pm 49pt$ in height and width.) As a result the resolution when METAFONT fails to compile our circular font is twice greater ($\approx$ 48 328 *dpi*) than it may be without such trick. (And it also enables to make the equivalent Type1 fonts not so big in discretes.)

The third problem is the following: what design size should be assigned to the font with the characters varying from $\pm 1pt$ to $\pm 49pt$? It is logically to select it to be 25*pt*, isn't it? — but the decision is not so simple. We need to take into account the POSTSCRIPT-technology — namely, the fact, that for Type1 fonts the design size corresponds to 1000 unscaled discretes, and that all coordinates characterizing the character should be integers (it is not obligatory, but desirable). So, the design size of 20*pt* is assigned for circle and bullet fonts — in this case the greatest elements are of order 2500 discretes, and the smallest elements (0.2*pt* in thickness) are of 10 discretes.

More thickness, more circles, more fonts

The typical line font, arrow font or circle font contains the elements of the same thickness. Like it was done in LATEX, for a different thickness there is a different font with the same encoding. (It is not true for bullet font which is not changed when the line thickness is changed by the User.) But while there are only *two* thicknesses in LATEX, PMPIC supports *four* of them: ordinary (`\thinlines`), thick (`\thicklines`), extra thick (`\Thicklines`) and extra thin (`\Thinlines`). As a result there are four line fonts and eight arrow fonts (as it was told in section *Lines and vectors from mosaic elements*, there are two arrow fonts with two arrow styles) — and it is essential that the arrow changes a little its size and style when line thickness is changed.

But if you think that there is just one bullet font and four circle fonts you are not correct — PMPIC uses *more* circlular fonts! In addition to the font with strictly circlular elements there are two additional fonts with elliptical elements — one for x-compressed circles, and one for y-compressed circles (in both cases the compression is done by factor 2) – it enables the User to make frames with elliptical corners, use elliptical elements for block-schemes, etc. So, there are *three* bullet bonts and *twelve* circle fonts finally.

It is an enormous amount of fonts, isn't it? Yes, may be. But not so many fonts as in European Computer Modern fonts, for example [11]. And these

Polska Grupa Użytkowników Systemu TEX, 2000 (http://www.GUST.org.pl)

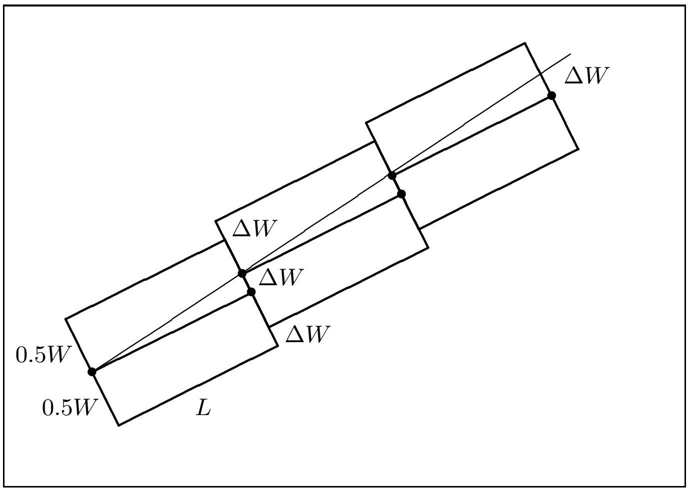

Figure 1: Overlapping of line segments with finite thickness and discrete slopes

fonts are downloaded only if the User tries to implement in a single document *all* graphical possibilities — PMPIC is intelligent enough to download font information only when it is required.

Line styles and vector styles

Up to now we talked about the graphical elements which extends the current LATEX power. Now it is time to discuss whan can be constructed from these elements.

First of all, it enables to draw lines and vectors. But even from LATEX elements it is possible to make more complex constructions than just a solid line [6, 7]. The package PMPIC follows this tradition — you can draw *more* than just a line with predefined and fixed style here. Vectors and lines are distinguished only formally — you can always add the arrow as the component of your line style. Next, you can add arrows at the beginning of the line with the same simplicity as to its end (or at both points). You can add a blank distance of a fixed length before the end point of your line or after its start point. (I.e., your line starts and finishes not exactly at specified points, but with some shift.) You can also add blank distance between the line itself and the bottom of the arrow attached to start or end line points. And you can also vary the style of the attached arrow: it can be of Lamport' style or Solomon' style, solid or outlined, normal or enlarged (totally 8 combinations).

The line style is also varied flexibely. You can select the style as the ordinary solid line and to vary its thickness (there are *four* predefined thickness sizes im PMPIC). You can make lines composed from *dots*, where nearly arbitrary symbol can be used as a *dot* and the distance between *dots* is also controlled. You can use solid lines with symbols placed over it at fixed distances (and if your symbol is 'transparent', it is not stroked by the solid line). You can also use dashed lines, where the dash length and the distance between dashes is defined by the User. Finally, you can select the dot-dashed style where the user-defined symbols are placed just in the middle of blank space between the dashes. There are several predefined styles for stroked, dotted, dashed and dot-dashed lines, and the User can vary the parameters of these styles flexibely.

It this all with lines? No, definitely not. For stroked, dotted, dashed and dot-dashed lines you can also control the alignment of the line with respect to its start and end points — i.e., your dashed line can start or end with a dash, or with a blank space, or with $^1/_2$ of a dash, etc. You can also use double and tripple lines where the distance between individual lines is under your total control. And, finally, you can change the way the individual lines for double and tripple lines are aligned with respect to each other — the parameter similar to line alignment, but independent from it. And, as a bonus, there are two special symbols which can be used as a *dot* for stroked, dotted and dot-dashed lines — these are the arrow heading aligned parallely to the line to be drawn and the tick aligned perpendicular to that line (with more than one style, as usual).

Circular elements

Circular elements are under development just now. It is planned that it is possible to draw circles and disks of various size, and also to draw the halves and

Polska Grupa Użytkowników Systemu TEX, 2000 (http://www.GUST.org.pl)

quaters of circles and disks for sizes where the corresponding elements are available. Additional circular elements available in PMPIC are the astro-shaped symbols and circular rings (filled or outlined). Similar to LATEX, oval boxes are available, but the variety of boxes and frames available in PMPIC should be more rich (see section *Boxes, frames, shadows*). In principle, the graphical primitives available in PMPIC fonts enable to draw quite exotic circular constructions — so, the User could train his/her fantasy doing it!

Additional graphical elements

While there should be some free cells in line and circle fonts available in PMPIC, there are no free cells there — they are used to draw special symbols suitable for stroked, dotted and dot-dashed lines and to put markers at user-defined points. Most of them are aligned in such a way, that the left baseline corner of its bounding box corresponds to its graphical center. Currently these symbols are:

`\arrow` — arrow heading with specified style, direction and alignment with respect current point,

`\tick` — tick (short solid line) with specified style and direction (tick is perpendicular to that direction),

`\picbbox` — black square

`\picwbox` — white square

`\picsquare` — small black square

`\picpoint` — small dot which size corresponds to line thickness

`\picdot` — small dot suitable for dash-dotted lines

`\picsnow` — 'snowflake'-like symbol

`\pictimes` — diagonal cross ($\times$)

`\picplus` — straight cross ($+$)

`\picbug` — combination of *plus* and *times* ($\ast$)

`\piccirc` — small circle with default size (it changes its size and thicknes when you change the line thickness)

`\piccirc*` or `\picdisk` or `\picbullet` — small bullet with default size (it changes its size when you change the line thickness)

`\picastro` — small astro-shaped symbol with default size

`\piccirc[`*size*`]` — small circle of specified size

`\piccirc*[`*size*`]` or `\picdisk[`*size*`]` — small bullet of specified size

`\picastro[`*size*`]` — small astro-shaped symbol of specified size

`\picbdiamond` — black 'diamond' symbol

`\picwdiamond` — white 'diamond' symbol

`\picbtriangleup` — black triangle-style symbol

`\picwtriangleup` — white triangle-style symbol

`\picbtriangledown` — black triangle-style symbol

`\picwtriangledown` — white triangle-style symbol

`\pictick` — tick, perpendicular to line to be drawn

`\picarrow` — arrow, parallel to line to be drawn

If you need to make your own symbol of that type (i.e., suitable as a marker or as a *dot* for for stroked, dotted and dot-dashed lines), you may use also the following commands:

`\pic{`*symbol*`}` — center the specified *symbol* with respect its height and depth (it is assumed that symbol is centered graphically with reprect its bounding box)

`\pic[`*size*`]{`*symbol*`}` — assign new *size* for the specified *symbol* (it is assumed that symbol is already centered by `\pic`)

`\pic*[`Δ*size*`]{`*symbol*`}` — increase or decrease the size of the specified *symbol* (it is assumed that symbol is already centered by `\pic`).

Boxes, frames, shadows

The predecessor of PMPIC — PMGRAPH [9] — enables to draw a lot of frame types with various corners, with single, double or tripple lines, and with sequence of black-and-white shadows attached to one of the corners of the frame. PMPIC extends this feature since it has much more graphical primitives suitable for frame and shadow drawing. This feature is under development now.

User-friendly syntax and turtle graphics

Quite limited subset of available graphical primitives is not the only disadvantage of the LATEX environment `picture`. The syntax of LATEX line drawing commands is also not perfect (see [6], for example). PMPIC enables the User to specify the lines, directions points, etc., in a very flexible manner. In particular, you can specify the end point of the line as

- point in absolute coordinates (x, y)
- point in relative coordinates $(\Delta x, \Delta y)$
- length and angle in degrees (L, α)
- length and direction as vector $(L, \Delta x, \Delta y)$
- point relative to current '*turtle position*'
- point relative to current '*origin position*'
- some or all of these values can be user-defined variables or calculated by the code embedded in your document — see below section *Arithmetics, variables, loops.* For example, you may specify the point by its symbolic name.

Polska Grupa Użytkowników Systemu TEX, 2000 (http://www.GUST.org.pl)

One of valuable additions to LaTeX syntax is the possibility to use *turtle graphics*. The package keeps in its mind the position and the direction of movement of the virtual pen (turtle). The User can rotate the turtle, to move the turtle to some other position and to draw with its help the lines by moving the turtle in specified direction by specified distance. The power of *turtle graphics* was already demonstrated in [12], for example.

Arithmetics, variables, loops

The disadvantage of LaTeX environment `picture` is that all the points and other drawing parameters are specified as constants. So, for example, when you need to modify slightly your picture, you generally need to modify a lot of code even if this transformation is as simple as shift, scaling or rotating. The package DRATEX [2] introduces the variables which enables to simplify this process greatly — and PMPIC follows its approach. Namely, there are numerical variables, point variables (pairs of values) and 'turtle position' variables (triples of values — point coordinates and angle), and there are the commands which enable the User to assign the values, to retrieve them and to manipulate these user-defined data.

For example, you can perform arithmetical calculations with your variables — addition, subtraction, multiplication, division and rounding. You can calculate the lengths and angles of vectors as well. Scalar and vector multiplications of vectors are also available. You can transform absolute lengths into relative values and backward using the current value of `\unitlength`. Finally, you can also measure the actual size of the circle to be drawn (circular elements, contrary to linear ones, have significant discrentness and you may wish to taske it into account carefully).

Specific but important component of these operations are geometrical calculations. You can

- find the point which is the intersection of two lines (you can define the line in various manner),
- select the point along the line connecting two points which divides the distance between these points in specified ratio,
- calculate the weighted sum of two vectors,
- make new point or new vector using the same flexibility as described above in section *User-friendly syntax and turtle graphics*,
- find the intersection of some line and the perpendicular passing through specified point,
- project the point over specified circle,
- find the points on the circle corresponding to tangentials passing through specified point outside the circle (absolute coordinates, polar coordinates),
- find the directions of the tangentials specified above,

You can also transform already defined points using one of the standard transformations available in PMPIC or defining the combination of standard transformations. These transformations include:

- shift by some vector,
- rotating with respect to some point,
- symmetry with respect to some point (rotating by 180°),
- inversion with respect to some circle,
- mirroring with respect to some line,
- general linear transformation (defined by its 6 coefficients).

As soon as there are variables, it is logical to have conditional operators and cycles as well. So they are in PMPIC. Although conditional operators already exist in TeX (built-in conditionals, IFTHEN LaTeX package, etc.), it is necessary to have some tools to combine them with PMPIC variables. So, there are macro commands enabling the User to analyze the sign of numerical value and to compare two values (the output is an ordinary TeX macro which can be used in TeX conditional commands).

If you are going to *program* your picture, it is natural to have loops as well. While it is possible to use TeX and LaTeX loops for this purpose, PMPIC allows you to use more simple loop — namely, the loop which just makes the cycle through the list of text elements (names, numbers, etc.). It just copies the functions of corresponding LaTeX loop except the fact that parasitic spaces inside your list are cleaned up. To make your life more happy there are also macro commands which generate the lists composed from integer values and real values — this feature helps to organize the cycle in a manner in is done in PYTHON language [13].

Except this explicit loop feature there are also *implicit* loops which help to make drawings composed from cyclic operations. Namely, there are the following commands (most of them are borrowed from EPIC.STY [6] with minor modifications):

`\multiput` — this command is copied directly from LaTeX `\picture` environment except the possibility to use variables (i.e., named scalar values and points) as its parameters,

`\matrixput` — just the same as `\multiput`, but with *two* independent incremental cycles,

Polska Grupa Użytkowników Systemu TeX, 2000 (http://www.GUST.org.pl)

`\multiputlist` — cycle over the list of text elements,

`\jput` — similar to `\put` but it also connects current point with the previous point by a line or a vector,

`\putfile` and `\jputfile` — cycles over pairs of coordinates (x, y) read from external file,

`\drawline` — connect by line segments the sequence of points.

Since it is much easier to use external program than TeX if you need really complicated calculations, there are no more sophisticated control structures in PMPIC.

Acknowledgements

We express our warmest thanks to our colleagues from other countries — their careful attention supports our activity in TeX even in such unfavourable conditions as they are in modern Russia.

References

[1] Michael J. Wichura. PICTeX: Macros for drawing PICtures. *TUGboat*, **9**, pp. 193–197 (August 1988).

[2] Eitan M. Gurari. TeX and LaTeX — Drawing and Literate Programming.

[3] Kristoffer H. Rose. Typesetting Diagrams with XY-pic: User's Manual. *Proceedings of the 7th European TeX Conference, Prague*, pp. 273–292, September 1992.

[4] Thomas E. Leathrum, Geoffrey Tobin. Pictures in TeX by METAFONT. Unpublished machine-readable document (supplement to MFPIC.STY), 1996.

[5] Denis Girou, Sebastian Rahts. Documentation for PSTricks package. Unpublished machine-readable document (supplement to PSTRICKS.STY), 1998.

[6] Sunil Podar. Enhancements to the picture environment of LaTeX (version 1.2). Unpublished machine-readable document (supplement to EPIC.STY), 1986.

[7] Conrad Kwok. EEPIC: Extensions to EPIC and LaTeX picture environment. Unpublished machine-readable document (supplement to EEPIC.STY), 1988.

[8] Ian Maclaine-cross. Curves in LaTeX pictures — a manual for CURVES.STY and CURVESLS.STY. Unpublished machine-readable document (supplement to CURVES.STY), 1995.

[9] A.S.Berdnikov, O.A.Grineva, S.B.Turtia. PMGRAPH: Some useful macros to extend LaTeX picture environment. *TUG'96 — Proceedings of the 17th Annual Meeting of the TeX Users Group, Dubna*, pp. 151–154, July 1996.

[10] David Solomon. Arrows for technical drawings. *TUGboat*, **13**, pp. 146–149 (July 1992).

[11] Jörg Knappen. European Computer Modern Fonts: the DC-fonts 1.3 — move towards stability and completeness. *TUG'96 — Proceedings of the 17th Annual Meeting of the TeX Users Group, Dubna*, pp. 8–10, July 1996.

[12] Kees van der Laan. Turtle Graphics and TeX — a child can do it. *TUG'96 — Proceedings of the 17th Annual Meeting of the TeX Users Group, Dubna*, pp. 128–131, July 1996.

[13] PYTHON home page: `http://www.python.org`.

◇ Olga A. Grineva
`olga@p50.f924.n5030.z2.fidonet.org`

◇ Galina V. Mitina
`Galina@p10.f108.n5030.z2.fidonet.org`

◇ Alexey V. Filippov
`alf@ptslab@ioffe.rssi.ru`

◇ Alexander S. Berdnikov
`berd@ianin.spb.su`

Polska Grupa Użytkowników Systemu TeX, 2000 (http://www.GUST.org.pl)