Graphics for TEX: a new implementation

Andrey V. Astrelin

Moscow
Russia
`astr@lcm.math.msu.su`

Graphic capacity is the branch unsufficiently developed in the basic TEX version. That leads to the large amount of the graphic packages for TEX. Some of them became standard (such as LATEX PICTURE package), but they are the most weak of all. The developed packages differ in the level of graphic information representation in the `.dvi` file. There are 3 levels:

1. The graphic information is represented via characters from some special fonts (like `line10` and `circle10` from LATEX). It may be done in two ways:
   - i. fixed fonts are created once and for all;
   - ii. each picture requires creating new fonts.
2. The graphic information is represented via `rule` commands (black rectangles). Example of this approach is the tables drawing (which is a form of graphics).
3. The graphic information is presented by the `\special` commands and processed by particular driver. Examples are `pcl` commands in PCTeX, line drawing and graphic files output in EmTeX and PostScript commands for the drivers generating PostScript output.

Selecting ways (1i) and (2), we have restricted opportunities for the picture creation, (the small set of the primitives requires a great number of them for the picture creation, that may result in TEX and printer memory overflow), and in (1ii) and (3) we have the compatibility problems.

For our implementation we chose the (3) level (`\special` commands usage) with the postprocessing of `.dvi` file.

In the low level of implementation the graphic command are represented in the form `\special{GR:commands}`, where `commands` are one or more command of the special language. Every command is represented by a character with some number arguments following it. During interpretation there are the following work variables:

"Points": one may keep up to $2^{15}-1$ points inside a page and use them for the spline drawing;

"Drawings": there are registers numbered from -2^{15} to $2^{15}-1$, containing drawings. Register 0 contains the current drawing and is an implicit argument in the most commands.

There are two types of drawings: paths and areas. Initially every drawing is constructed as a path, i.e. set of splines. One may add a spline to the current path, transform a path to an area by the `draw` command, that is to draw a path with the pen `N`, or implicitly convert a set of closed paths to an area (like the METAFONT command `fill` do).

The drawings in the registers with positive indices are fixed in the page: when we use them as arguments of the command `load` or area commands (see below), they remain in the same place of the page where they were drawn, and the registers with the negative indices contain the replaceable drawings: the reference point where the `save` command was executed is the base point of drawing, and if we restore it at a different point, it moves to a corresponding vector.

Below in the list of graphic commands:

e erase drawing 0;
g go to the reference point and use it as the start base point of the next spline;
c use the reference point as the control point of the current spline;
b use the reference point as the end base point of the current spline and as the start base point of the next one;
tN assign the reference point coordinates to the point register `N`;
@N execute the following commands using the point `N` as reference point;
sN save the current drawing in the register `N`;
lN load the current drawing from the register `N`;
fN,a,b,c,d transform drawing with the matrix `((a/N b/N) (c/N d/N))`;

Path commands:

dN draw the current path with the pen from register `N` (`N` should be negative);
hN,N1,N2,... replace a path with a hatch; `N1/N,N2/N,...` are the parts of path length, that are included/not included in the hatch;
HN1,N2,... replace a path with a hatch; `N1,N2,...` are the lengths (in `sp`) of path segments that are included/not included in the hatch;

Area commands:

+N create an union of the current drawing and the drawing contained in the register `N`;

*N create an intersection of the current drawing and the drawing contained in the register `N`;

-N subtract the drawing contained in the register `N` from the current drawing;

_ output the drawing 0 to `.dvi` file.

The `.dvi` file with the commands mentioned above is processed by the special program and post processor. As a result we get new `.dvi` file in which all the graphic `\special`'s are removed and some commands `\special{em:graph $$NNNN.pcx}` are inserted. Also some graphic files are created.

The second level of the product is a set of TEX macros that may be used to place the graphic `\special`'s to the proper places of the page. They are written in the way to reduce the `.dvi` file (and TEX memory) size. There are also some macros for placing TEX boxes into the drawings.

The third level is the library of graphic macros for users. Now it is not fully developed and we shall not describe it.

In future we plan to use some alternative forms of the graphic output. Among them are automatic generation of `.pk` fonts; and generation of PostScript output. Also there are some ideas of replacing the `\special`'s form of graphic information by the direct output to the text file for postprocessing, but in that way we shall lose the possibility of usage of TEX-dependent points as spline base or control points, and there will be some problems with graphic representation: if we include the graphic file at some point, we may lose top and left sides of picture and if we use characters of a font to be created, we need multiple passes of composing.