Graphics

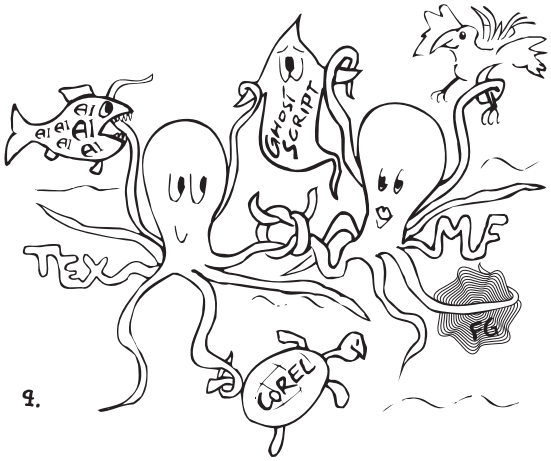
A METAFONT–EPS interface

Bogusław Jackowski

Do not explain too much.
W. Strunk Jr. and E. B. White,
The Elements of Style

Introduction

TEX is not a lion, TEX is an octopus... This sounds like heresy, but it is my deepest conviction that one of the most wonderful features of the TEX/METAFONT system is its openness, i.e., the capability of collaboration with other systems. Hence the association with an octopus:



The paper illustrates this statement by presenting a brief description of an interface for METAFONT-to-PostScript, MFTOEPS. The kernel of the package is a METAFONT program (`MFTOEPS.MF`) which provides necessary definitions for translating the description of graphic objects from METAFONT to PostScript. The PostScript code is written to a log file. It can be extracted from the log file either manually or with the help of additional utilities. There are two programs in the package for performing this task: an AWK program and a TEX program, the latter a bit slower but more universal.

The PostScript files (specifically, Encapsulated PostScript files) produced by MFTOEPS are readable by some popular graphics programs, namely, by Adobe Illustrator (Macintosh and PC compatibles), CorelDraw! (PC compatibles), and Fontographer (Macintosh and PC compatibles). In other words, graphic objects programmed using METAFONT can be further processed by these programs.

It should be stressed that it not the idea of employing METAFONT to produce PostScript code that is important here. A much better tool for this purpose is J.D. Hobby's METAPOST. It is possibile to further process the objects generated by MFTOEPS that makes this package worthy of mention.

Overview of the MFTOEPS package

The `MFTOEPS.MF` program contains the definitions of the following macros which are meant to be used for generating EPS files:

```
eps_mode_setup    fix_line_width
write_preamble    fix_line_join
write_postamble   fix_line_cap
find_BB           fix_miter_limit
set_BB            fix_dash
fill_C            fix_fill_cmyk
draw_C            fix_draw_cmyk
clip_C
```

Obviously, not all possibilities of PostScript are exploited, but the main idea was to provide a *simple tool* for producing output "eatable" by programs which are not PostScript interpreters. Therefore only a small subset of the PostScript language can be taken into account. Nevertheless, these 15 commands are enough to produce an innumerable variety of graphic objects.

METAFONT programs using MFTOEPS have the following structure:

```
1. input mftoeps;
2. eps_mode_setup; % instead of mode_setup
3. ⟨ METAFONT code ⟩
4. find_BB ⟨ list of paths ⟩;
5. write_preamble jobname;
6. ⟨ METAFONT code containing fill_C, draw_C,
     clip_C, etc. ⟩
7. write_postamble;
8. end.
```

The structure seems straightforward, except for some notational details which will be explained in a moment. Perhaps only the fourth line needs a few remarks. A properly formed EPS file should contain the coordinates of the corners of the bounding box in a comment line at the beginning of the file. Macro `write_preamble` needs to know the respective coordinates, as it is responsible for generating the header of an EPS file. Macro `find_BB` simply prepares the data for `write_preamble`.

As you can see, using the plain `beginchar` and `endchar` commands is not essential, although usually it is convenient to make use of them.

Synopsis of the interface of the MFTOEPS package

Conventions: In the following I shall use the words *number*, *pair*, *string*, and *path* as abbreviations for *numeric expression*, *pair expression*, *string expression*, and *path expression*, respectively. The angle brackets, ⟨ and ⟩, used for marking parameters of macros, are "meta-characters," i.e., they do not belong to the METAFONT code.

COMMAND:
`eps_mode_setup`

USAGE:
`eps_mode_setup` ⟨ *an optional number (0 or 1)* ⟩`;`

REMARKS:
This command should be used *instead of* the usual `mode_setup` command. The forms `eps_mode_setup` and `eps_mode_setup 1` are equivalent. One of them (preferably the former) should be used for normal processing, i.e., for generating EPS files. Invoking `eps_mode_setup 0` is meant primarily for testing purposes and is supposed to be used by experienced programmers who know what they are doing. There is a string variable, `extra_eps_setup`, similar to `extra_mode_setup`; at the end of `eps_mode_setup` the command `scantokens extra_eps_setup` is invoked, enabling user-oriented adjustments, e.g., changing the default resolution.

COMMAND:
`write_preamble`

USAGE:
`write_preamble` ⟨ *string* ⟩`;`

REMARKS:
This command initializes the process of writing the PostScript code. The string expression is the name (without extension) of the resulting EPS file; the extension is always `EPS`. METAFONT is switched to `batchmode` in order to avoid slowing down the process by writing mess(ages) to the terminal. Inspection of the log file is thus highly recommended.

COMMAND:
`write_postamble`

USAGE:
`write_postamble;`

REMARKS:
This command ends the writing of the PS code, switches METAFONT back to `errorstopmode`, and performs necessary "last minute" actions (see below).

COMMANDS:
`set_BB find_BB reset_BB`

USAGE:
`set_BB` ⟨ *four numbers or two pairs separated by commas* ⟩`;`
`find_BB` ⟨ *a list of paths separated by commas* ⟩`;`
`reset_BB;`

REMARKS:
The commands `set_BB` or `find_BB` should be invoked prior to invoking `write_preamble`. `set_BB` sets the coordinates of the corners of the bounding box of a graphic object; it is useful when the bounding box of a graphic object is known in advance or if it is required to force an artificial bounding box. `find_BB` computes the respective bounding box for a list of paths; if several `find_BB` statements are used, the common bounding box is calculated for all paths that appear in the arguments. The result is stored in the variables `xl_crd`, `yl_crd`, `xh_crd`, and `yh_crd`. There are two functions, `llxy` and `urxy`, returning pairs `(xl_crd,yl_crd)` and `(xh_crd,yh_crd)`, respectively. The last command, `reset_BB`, makes `xl_crd`, `yl_crd`, `xh_crd`, and `yh_crd` undefined (the initial situation); `reset_BB` is performed by the `write_postamble` macro, which is convenient in the case of generating several EPS files in a single METAFONT run.

COMMANDS:
`fill_C draw_C`

USAGE:
`fill_C` ⟨ *a list of paths separated by commas* ⟩`;`
`draw_C` ⟨ *a list of paths separated by commas* ⟩`;`

REMARKS:
These commands are to be used instead of the usual METAFONT `fill` and `draw` ones. They cause a list of paths followed by the PostScript operation `eofill` (`fill_C`) or `stroke` (`draw_C`) to be translated to a PostScript code. The list of paths constitutes a single curve in the sense of PostScript.

COMMAND:
`clip_C`

USAGE:
`clip_C` ⟨ *a list of paths separated by commas, possibly empty* ⟩`;`

Remarks:
The macro `clip_C` with a non-empty parameter works similarly to the `fill_C` command, except that the `eoclip` operator is issued instead of `eofill`. This causes an appropriate change to the current clipping area. According to PostScript's principles, the resulting area is a set product of the current clipping area and the area specified in the argument of the `eoclip` command. The empty parameter marks the end of the scope of the most recent `clip_C` command with a non-empty parameter. In other words, nested `clip_C` commands form a "stack" structure. If needed, the appropriate number of parameterless `clip_C` commands is issued by the `write_postamble` macro, thus the user does not need to worry about it. Warning*: files produced using* `clip_C` *are interpreted properly by Adobe Illustrator (provided path directions are defined properly) but not by CorelDraw! (ver. 3.0).*

Commands:
`fix_line_width fix_line_join`
`fix_line_cap fix_miter_limit`
`fix_dash`

Usage:
`fix_line_width` ⟨ *a non-negative number (dimension)* ⟩`;`
`fix_line_join` ⟨ *a number (0, 1 or 2)* ⟩`;`
`fix_line_cap` ⟨ *a number (0, 1 or 2)* ⟩`;`
`fix_miter_limit` ⟨ *a number* ≥ 1 ⟩`;`
`fix_dash` (⟨ *a list of numbers (dimensions) separated by commas, possibly empty* ⟩) ⟨ *a number (dimension)* ⟩

Remarks:
These command are to be used in connection with the `draw_C` command. `fix_line_width` fixes the thickness of the outline. The other four commands correspond to PostScript operations `setlinejoin`, `setlinecap`, `setmiterlimit`, and `setdash` (see the *PostScript Language Reference Manual* for details). All commands should be used after `write_preamble`, as `write_preamble` sets the default thickness (0.4pt), default line join (0), default line cap (0), default miter limit (10), and a solid line as a default for stroking (`fix_dash () 0`).

Commands:
`fix_fill_cmyk fix_draw_cmyk`

Usage:
`fix_fill_cmyk` ⟨ *four numbers separated by commas* ⟩`;`
`fix_draw_cmyk` ⟨ *four numbers separated by commas* ⟩`;`

Remarks:
These commands define the colours of the interiors of graphic objects (`fix_fill_cmyk`) and colours of outlines (`fix_draw_cmyk`) using the cyan-magenta-yellow-black model (the basic model of the MFTOEPS package). They should be used after `write_preamble` (because `write_preamble` defines the black colour as a default for both macros) and prior to invoking the corresponding `fill_C` and `draw_C` commands. There are also (just in case) macros `fix_fill_rgb` and `fix_draw_rgb` using red-green-blue model; the argument to both of these macros is a triple of numbers. (The user can control the process of conversion from RGB to CMYK by the redefinition of macros `under_color_removal` and `black_generation`.) The numbers forming the arguments of the macros are supposed to belong to the interval [0...:1].

Besides the fifteen basic macros there are two functions and two control variables that may be of some interest for a virtual user of the MFTOEPS package:

Additional functions:
`pos_turn neg_turn`

Usage:
`pos_turn (`⟨ *path* ⟩`)`
`neg_turn (`⟨ *path* ⟩`)`

Remarks:
Each function returns the path passed as the argument, except that the orientation of the path is changed, if necessary: `pos_turn` returns paths oriented counter-clockwise, `neg_turn` — oriented clockwise. This may be useful for creating pictures which are to be processed further by Adobe Illustrator, because this program is sensitive to the orientation of paths.

Control variable:
`yeseps`

Remarks:
No EPS file will be generated unless the variable `yeseps` is assigned a definite value. It is advisable to set this variable in a command line (see section "Examples").

Control variable:
`testing`

Remarks:
If the variable `testing` is assigned a definite value, the whole PostScript code is flushed to the terminal, thus slowing down significantly the process of generating an EPS file (cf. the description of the `write_preamble` command).

Examples

All sample programs in this section are presented *in extenso*. The reader is not supposed to study the code thoroughly. Nevertheless, I prefer to leave the reader to decide which parts of the code are to be skipped.

Let us start with a trivial example of a "pure" METAFONT program:

```
1. beginchar(48, % ASCII code
2.  2cm#, % width
3.  1cm#, % height
4.  0cm#  % depth
5.  );
6.  fill unitsquare xscaled w yscaled h;
7. endchar;
8. end.
```

The program, obviously, generates a font containing one character: a darkened rectangle $2\,\mathrm{cm} \times 1\,\mathrm{cm}$. In order to generate an EPS file containing the same figure, a few modifications are necessary:

```
1. input mftoeps;
2. eps_mode_setup;
3. beginchar(48, % just something
4.  2cm#, % width
5.  1cm#, % height
6.  0cm#  % depth
7.  );
8.  set_BB 0,-d,w,h; % coordinates
9.                   % of the corners
10.                  % of the bounding box
11.  write_preamble "rectan";
12.  fill_C unitsquare xscaled w yscaled h;
13.  write_postamble;
14. endchar;
15. end.
```

Four new commands have appeared: `eps_mode_setup`, `set_BB`, `write_preamble` and `write_postamble`; moreover, `fill` has been replaced by `fill_C`. This is a usual routine for converting an "ordinary" METAFONT program to a form suitable for generating EPS files. Obviously, `draw` should be replaced by `draw_C`, and `filldraw` — with the two operations `fill_C` and `draw_C`. In the latter case the order of the operations `fill_C` and `draw_C` is significant if the drawing and filling colours are different.

Having made this change you can easily generate the respective EPS file, provided you are a DOS user. Assume that the modified program is stored in the file `RECTAN.MF`. In the package MFTOEPS you will find a DOS batch, `M2E.BAT` (subdirectory `PROGS`), which — perhaps after slight adjustments — can be used for this task. It is enough to write

```
m2e rectan
```

(no extension, please) from the command line in order to obtain the required `RECTAN.EPS` file. The batch makes use of AWK for extracting the PostScript code from the log file. There is also an alternative batch, `M2E-ALT.BAT`, that employs TEX for this purpose. In both batches METAFONT is called in the following way:

```
mf386 &plain \yeseps:=1; input %1
```

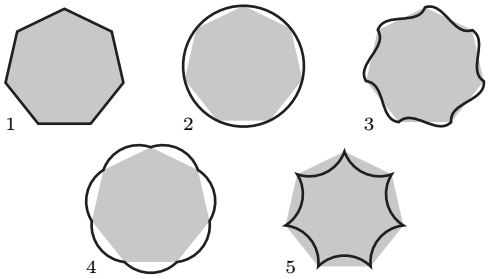
Observe the assignment `yeseps:=1`. In fact, assigning a definite (arbitrary) value to the `yeseps` variable triggers the action of generating an EPS file.

I hope that making scripts for other operating systems is not found to be extremely difficult. I would be very much obliged if others could contribute such scripts to the package.

Let us consider now a more complex example. Suppose that the file `POLYGON.MF` contains the following definitions:

```
1. vardef regular_polygon(expr n) =
2. % n is the number of vertices;
3. % the diameter of the circumscribed
4. % circle is equal to 1, its centre
4. % is in the origin
5.  (up % first vertex
6.   for i:=1 upto n-1:
7.    --  % next vertices:
9.    (up rotated (i*(360/n)))
9.   endfor
10.  -- cycle) scaled .5
11. enddef;
12. vardef flex_polygon(expr n,a,b) =
13. % n is the number of vertices,
14. % a, b are the angles (at vertices)
15. % between a tangent to a ``flex side''
16. % and the corresponding secant
17.  save zz;
18.  pair zz[ ]; % array of vertices
19.  for i:=0 upto n-1:
20.   zz[i]:=up rotated (i*(360/n));
21.  endfor
22.  (zz[0] {(zz[1]-zz[0]) rotated a}
23.   for i:=1 upto n-1:
24.    .. {(zz[i]-zz[i-1]) rotated b}
25.    zz[i]
26.    {(zz[(i+1) mod n]-zz[i]) rotated a}
27.   endfor
28.   .. {(zz[0]-zz[n-1]) rotated b} cycle)
29.  scaled .5
30. enddef;
```

The first function, `regular_polygon`, returns a closed path that is — as the name suggests — a regular polygon with a given number of vertices. The second function, `flex_polygon`, returns a curve that is in a sense a "generalized polygon" — the following examples show why this epithet is appropriate:



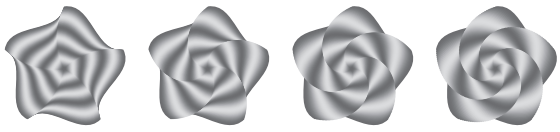
The first picture was generated by the following program:

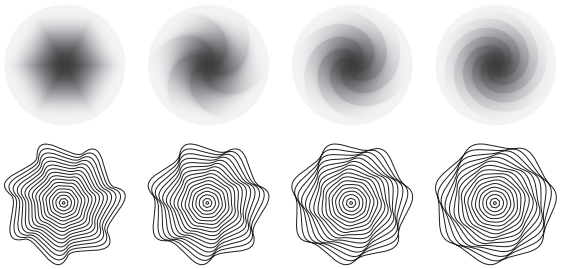
```
1. input polygons;
2. input mftoeps;
3. eps_mode_setup;
4. beginchar(0,16mm#,16mm#,0);
5.  path P[ ]; % ``room'' for two polygons
6. % preparing:
7.  P[1]:=regular_polygon(7)
8.   scaled w shifted (.5w,.5h);
9.  P[2]:=flex_polygon(7,0,0)
10.   scaled w shifted (.5w,.5h);
11. % exporting:
12.  find_BB P[1], P[2];
13.  write_preamble jobname;
14. % 25 percent of black for filling:
15.  fix_fill_cmyk 0,0,0,.25;
16.  fix_line_width 1pt;
17.  fill_C P1; draw_C P2;
18.  write_postamble;
19. endchar;
20. end.
```

The remaining four figures can be obtained by simple modifications of line 9 of the program:

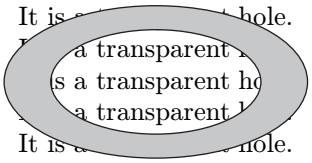
```
P[2]:=flex_polygon(7,-180/7,180/7)  % 2
P[2]:=flex_polygon(7,45,45)         % 3
P[2]:=flex_polygon(7,-45,45)        % 4
P[2]:=flex_polygon(7,45,-45)        % 5
```

These fairly trivial objects can be used to achieve some rather non-trivial effects (METAFONT sources are included in the MFTOEPS package):





So far the examples have contained `fill_C` and `draw_C` commands with arguments being single paths. PostScript, in contrast to METAFONT, accepts groups of paths as a single curve. Therefore the `fill_C` and `draw_C` commands were defined to accept the lists of METAFONT paths as arguments. In the resulting PostScript code they constitute a single object. The main reason is that such objects may contain transparent holes. This enables achieving such effects as:



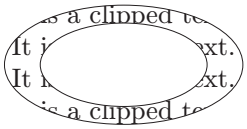
The graphic object was generated by the following simple program:

```
1. input mftoeps; eps_mode_setup;
2. w#=4cm#; h#=2cm#; define_pixels(w,h);
3. set_BB origin, (w,h);
4. write_preamble jobname;
5. % 25 percent of black for filling:
6.  fix_fill_cmyk 0,0,0,.25;
7.  fix_line_width 1pt;
8.  for oper:="draw_C", "fill_C":
9.  scantokens oper
10. % outer edge:
11.   fullcircle
12.    xscaled w yscaled h
13.    shifted (.5w,.5h),
14. % inner edge:
15.   reverse fullcircle
16.   xscaled .7w yscaled .7h
17.   shifted (.5w,.5h);
18.  endfor
19. write_postamble;
20. end.
```

One innocent trick was used in order to shorten the code: the loop in the combination with the `scantokens` command (lines 8 and 9). It is advisable to have paths that form transparent holes appropriately oriented — therefore the operator `reverse` is used in line 15. A TEX code for obtaining the above figure is obvious: it is enough

to put the picture on top of a text box, using, for example, the `\llap` command.

Removing the command `fix_fill_cmyk` (line 6) and replacing the command `fill_C` (line 8) by `clip_C` gives the opportunity to obtain yet another effect:



In this case, however, the TEX code is somewhat complicated, since macros for inclusion of an EPS file (I use Tomas Rokicki's `EPSF.TEX`) embed the code of the EPS file into a PostScript `save`–`restore` group. A clipping path is subjected to such a grouping, contrary to the state of the currently painted picture. Therefore some `\special` hackery is needed (the respective TEX source is included with samples in the MFTOEPS package).

The distinction between single and multiple paths in the context of drawing outlines (`draw_C`) is meaningless.

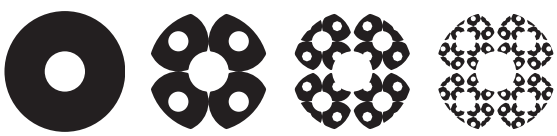
The final example shows how to use clipping to generate a geometric figure known as "Sierpiński's carpet". In order to construct the "carpet" you start with a square with a central hole; this hole is a square with each edge one-third the length of the edge of the original square. Now you divide the original figure into nine squares and replace all filled small squares with a copy of the square with the central hole, scaled down to fit the area of the small square. Then you apply the same procedure to the smaller squares, an so on, *ad infinitum*.

Here you have the program accomplishing this task (infinity "equals" three):

```
1. input mftoeps; eps_mode_setup;
2. % ---
3. def ^ = ** enddef; % syntactic sugar
4. primarydef i // n = % ditto
5.  (if n=0: 0 else: i/n fi)
6. % why not to divide by 0?
7. enddef;
8. def shifted_accordingly(expr i,j,n,D)=
9.  shifted ((i//n)[0,w-D],(j//n)[0,w-D])
10. enddef;
11. % ---
12. w#=16mm#; h#=16mm#; define_pixels(w,h);
13. for N:=1,2,3: % 4, 5, 6, ..., infinity
14.  set_BB 0,0,w,h;
15.  write_preamble jobname & decimal(N);
16.  D:=3w;
17.  for n:=
18.   0 for q:=1 upto N-1: , 3^q-1 endfor:
19. % i.e.:
20. % ``for n:=0, 3^1-1, ..., 3^(N-1)-1:''
21.  path p[], q[]; D:=1/3D; k:=-1;
22.  for i:=0 upto n: for j:=0 upto n:
23.   k:=k+1;
24.   p[k]=unitsquare scaled D
25.    shifted_accordingly(i,j,n,D);
26.   q[k]=reverse unitsquare scaled 1/3D
27.    shifted (1/3D,1/3D)
28.    shifted_accordingly(i,j,n,D);
29.  endfor; endfor;
30.  clip_C p0, q0
31.   for i:=1 upto k:
32.    , p[i], q[i]
33.   endfor;
34. endfor;
35. fill_C unitsquare scaled w;
36. write_postamble;
37. endfor;
38. % ---
37. end.
```

The program is lengthy mainly because of technical details that are not especially interesting; however, there are three points worthy of comment. First, observe that a couple of EPS files are produced in one METAFONT run (the loop in line 13 is relevant here); second, loops are used to form arguments to the loop in line 18 and to the `clip_C` command in line 30 — it is a very useful feature of METAFONT that loops behave exactly like macros; and third, observe that the operation `fill_C` is used only once. The resulting EPS files are shown in the following picture:

You may argue that such a figure can be generated easily in a simpler way, without clipping. True, yet I like this approach — can you imagine a straightforward method for generating a "circular carpet" without clipping? Moreover, one can use



clipping in more complicated situations, not only for filling. But, on the other hand, finding the precise bounding box for a clipped figure becomes a non-trivial task. You must remember, moreover,

that clipping consumes a lot of the resources of a PostScript interpreter, thus it should be used with great care.

Final remarks

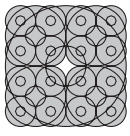
The MFTOEPS package was not devised as a competitor to such giants as Adobe Illustrator or CorelDRAW!. On the contrary, it can be regarded as their little ally. Interactive programs don't cope particularly well with tasks that bear logical structure. In such cases METAFONT — with its wealth of *programmable* path operations, absent "by definition" from the menus of interactive programs — is certainly a preferable tool.

One of the advantages of the applied approach is its portability — the only software needed is METAFONT and either AWK or TEX. Another advantage is its flexibility. It is not particularly difficult to modify the MFTOEPS package to produce another PostScript dialect, if for some reason the dialect of Adobe Illustrator is inconvenient. MFTOEPS can also be modified to produce output in other lingos, e.g., HP-GL (Hewlett-Packard Graphic Language).

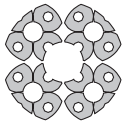
There is still a lot of work to be done. Of course, every program can be improved, but perhaps more important would be preparing a library of METAFONT routines useful for creating objects with a vector representation.

For example, it would be convenient to have a procedure which, for a given set of graphic objects finds a single curve (outline) filling of which would give the same optical result. In other words, such a procedure would perform the task of finding an outline for a set union of graphic objects. Such a procedure is known as *removing overlaps*. The example of the "circular carpet" (see above) illustrates a similar problem: to find an outline for a set intersection of a group of graphic objects.

If the carpet is generated using clipping, the PostScript file contains, in fact, the following elements:



They are partially invisible because of clipping, still they are there. In some contexts, e.g., if the figure is to be cut on a cutting plotter, it is crucial to replace such a multiplicity of objects by a single object:



Note that routines for finding the outline of a set union or a set intersection of a group of graphic objects are not MFTOEPS-oriented. A package providing tools for programming such operations, ROEX, is already available. Perhaps it is most useful in the context of exporting to EPS, however, it can be used with plain METAFONT, and — I guess — with METAPOST as well.

Universal routines of this kind are important from the point of view of the openness of the TEX/METAFONT system, and its openness — as was already mentioned — is one of the most powerful features of the system.

Note also that the openness of a system concerns both *output* and *input*. MFTOEPS accomplishes the first part of the conjunction, but one can think also about an export from PostScript to METAFONT. A package accomplishing this task, `PS_CxONV`, has been recently released as a public domain contribution. Its kernel is a converter, written in PostScript and using the GHOSTSCRIPT interpreter of PostScript, which translates a general PostScript code into a canonical EPS form (there exists a similar program in the standard GHOSTSCRIPT distribution, namely, `PS2AI`, written by Jason Olszewski, but it does not fit this particular problem); the result of such a conversion can be translated to a METAFONT program using the AWK-based utility, EPSTOMF, also recently released into the public domain. This would complete a link between METAFONT and PostScript. I do believe that providing such links is one of the most efficient routes towards a limitless development of the TEX/METAFONT system.

Glossary

AWK: a simple yet powerful batch text processor.

Bounding box: the smallest rectangle surrounding the glyph of a picture; coordinates of its lower left and upper right corners (in big points) should appear in a structural comment in a header of an EPS file.

EPS file: Encapsulated PostScript file; a single-page PostScript document; the purpose of the EPS file is to be included ("encapsulated") as a part of other PostScript programs and to exchange graphic data among applications.

Even-odd rule: a rule that specifies the interior of a (multiple) path in the following way: if

for a given point and for any ray drawn from this point to infinity, the number of intersection points of the ray and the path is odd, the point is inside; if the number is even, the point is outside; command `eofill` and `eoclip` operators follow this rule.

Path orientation: nodes of a closed single path are ordered; if traversing a path following the order of its nodes results in a counter-clockwise turn(s), the path is positively oriented, if it results in a clockwise turn(s), its orientation is negative; the number of turns (signed) is called a turning number (METAFONT) or a winding number (PostScript); the operators `fill` and `clip` make use of a winding number, the operators `eofill` and `eoclip` ignore it.

Availability

The packages MFTOEPS, EPSTOMF and `PS_CONV` can be found at

`ftp.pg.gda.pl`

in the directories

`/pub/TeX/GUST/contrib/MF-PS/MFTOEPS`
`/pub/TeX/GUST/contrib/MF-PS/EPSTOMF`
`/pub/TeX/GUST/contrib/PS/PS_CONV`

References

[1] Adobe Systems Inc. *PostScript Language Reference Manual*. Reading, Mass.: Addison-Wesley, 1991.

[2] Aho, A.V., B.W. Kernighan, P.J. Weinberger. *The* AWK *Programming Language*, Reading, Mass.: Addison-Wesley, 1988.

[3] Jackowski, B., M. Ryćko. "Labyrinth of METAFONT paths in outline." *Proceedings of the Eighth European TEX Conference* (Sept. 26–30, 1994, Gdańsk, Poland), pages 18–32.

[4] Knuth, D.E. *The* METAFONT*book*. Reading, Mass.: Addison-Wesley, 1992. D. E. Knuth: *The* METAFONT*book*, Addison-Wesley, 1992.

◇ Bogusław Jackowski
BOP s.c., Gdańsk, Poland