

Paradigms: Just a little bit of PostScript

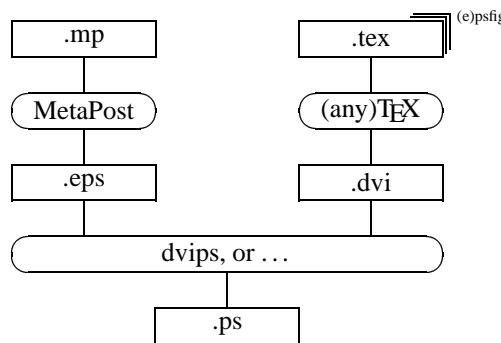
Kees van der Laan

Abstract

It is all about creating EPS—with graphics—to be merged with (La)TeX scripts. The emphasis is on creating raw PostScript for simple symmetrical pictures. Asides, like incorporating accurate graphs of math functions, typesetting text along curved paths, or tables set sideways, next to reverse video, clipping and tiling have been addressed. A poor man's *mftoeps* approach is touched upon: (declarative) METAFONT into (imperative) PostScript.

1 BLUe's Design X

Hi folks. The user's guide which comes with BLUe's format system—Publishing with TeX, PWT for short—is processed *completely* by TeX, *no* other tools such as POSTSCRIPT are needed.¹ However, of late I exercised METAFONT—well, eventually MetaPost with the help of Jos Winnink—for graphics to be included in TeX documents, and finally embarked PostScript straightaway to create EPS pictures, with the help of Joseph Romanovsky.



POSTSCRIPT is involuntary needed to (electronically) paste up the graphics, and as resulting file format.²

If we come to think of graphics as

just doing the 'right' strokes or fills

¹Nobody knows what the future has in store, but for the moment I consider it a good thing that the PWT guide can be processed just by TeX, well ... with BLUe's format.

²For exchange the .tex and (hand-coded) .eps files are much better suited because of their conciseness. This can't be beaten, not even by Adobe's PDF—Portable Document Format.

³PostScript II also provides for colors and processing in a network.

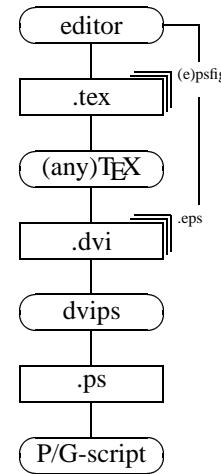
⁴Another way for arriving at the EPS code is to use Jackowski's *mftoeps* package or to use MetaPost.

⁵See Bentley's Little languages in 'More programming pearls—Confessions from a coder.' Addison-Wesley.

then POSTSCRIPT provides the means for this: lines, splines and circular arcs, to be drawn or filled.³ I use the sidestep

METAFONT → MetaPost → EPS

for general pictures but also for obtaining the right (control) points explicitly from a declarative specification in METAFONT, as shown by Escher's knot at the end.⁴



With respect to graphics POSTSCRIPT can be seen as a *little* language in the UNIX tradition.⁵ A little bit of POSTSCRIPT adheres the 80%–20% adage: 80% of the effects (or more) with 20% of the energy (or less).

One can with a little knowledge of POSTSCRIPT code graphics immediately and *completely* in POSTSCRIPT. The more so because of the ubiquitous public domain GhostScript previewers to verify the result, next to of course the POSTSCRIPT laser printers.

Furthermore, text is just a special case of graphics, and merging just a little bit of text—malenki Russians would say—with the graphics goes equally simple at first glance.

And to end the lovesong the inclusion of accurate graphs of mathematical functions goes well via coding these in

POSTSCRIPT and including these as figures. (Of course Hobby's graph extension could be used as well, or other advanced graphics packages.) This is illustrated by a graph of the sine function to convey the idea.

PStricks is about *interfacing*. Not assuming knowledge of POSTSCRIPT. This note discusses mainly *merging*. Is about extending your \TeX expertise with just a little—tsjut-tsjut Russians would say—knowledge of POSTSCRIPT rewarded by high returns.

Below I'll summarize what is needed from POSTSCRIPT, and illustrate the use of it with a few examples, introducing en route the operators we need given the example.

2 PostScript

2.1 Processing

POSTSCRIPT comes with a user's guide (cookbook) and reference manual, the so-called blue and red books in the Adobe POSTSCRIPT series. For processing POSTSCRIPT an interpreter is needed, such as a POSTSCRIPT laser printer or a GhostScript previewer. For inclusion in (La) \TeX I use the psfig macros.⁶ Goossens in his PostScript and (La) \TeX , MAPS 92.1, nicely details about inclusion of PostScript.⁷

As with PDF I consider the post-processing capability *independently* from the tool which created the POSTSCRIPT source, very powerful and flexible.

2.2 Why writing PostScript?

History has it that POSTSCRIPT programs are not written by humans but generated by high-level tools, such as MetaPost. This is understandable given the low-level nature of POSTSCRIPT. However, it is feasible to write ‘little’ POSTSCRIPT programs where use is made of the graphic primitives to perform the right graphic strokes, with little effort and high gains. The red and blue books don't provide the simplest examples—they illustrate the power of POSTSCRIPT—the codes are frightening and might put you off. Maybe my backside of the envelope codes will persuade you to try some gems of your own.

But, ... the proofing is cumbersome still, alas. This can be counteracted by a discipline of POSTSCRIPT coding, hopefully⁸.

2.3 The audience

This paper is aimed at users of (La) \TeX who agree with me that graphics has all to do with the right strokes. Once we know those it should be a simple coding problem to draw these strokes by POSTSCRIPT. The latter is explained

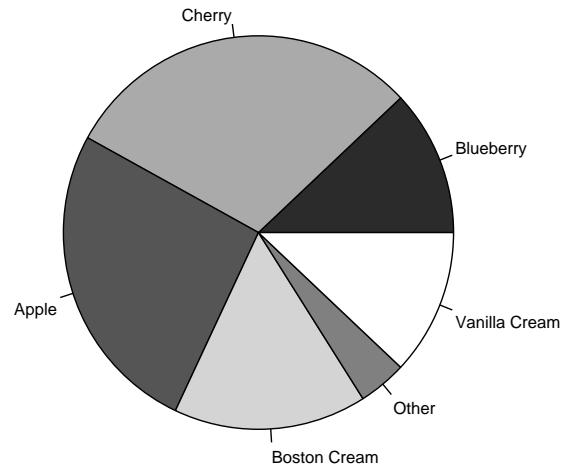
along with simple examples, which are prompted by generally required functionalities.

On the other hand, as communicated by Nicolaj Tretjakov, this paper might not be that relevant, because people in practice receive POSTSCRIPT files along with some representation of a script to coerce it into a (translated) book or so. I don't know how to address that audience, as yet.

2.4 Documentation

The red book—the reference manual—is generally recommended, though the blue book—the tutorial and cookbook—is also nice.⁹

Example (Pie chart from the blue book)



January Pie Sales

The invoke is essentially as follows and shows that the codes can be used straightforwardly.¹⁰ It is no longer necessary to mesh around with the picture environment or so, to achieve the effect.

```
%preliminary matter
(January Pie Sales)
24 12 %... array size
[ [ (Blueberry) .12 ]
  [ (Cherry) .30 ]
  [ (Apple) .26 ]
  [ (Boston Cream) .16 ]
  [ (Other) .04 ]
  [ (Vanilla Cream) .12 ]
] 306 396%translate center to
    140      %size
    DrawPieChart
showpage
```

⁶Courtesy Trevor J. Darrell.

⁷See also the L^AT_EX Companion.

⁸Well, professionally there is no other way then to resort on the high-level tools.

⁹I used the POSTSCRIPT I red book and this is well-suited to get the flavour. For \TeX and METAFONT this is similar. To grasp the basic ideas Knuth's first book is a more concise survey of the main lines of thought than The \TeX book and the METAFONTbook.

¹⁰It is not standard POSTSCRIPT. We have to construct some kind of library to use the PostScript programs from. Maybe the CTAN as global network library? Copied on the various CD-ROMS?

2.4.1 PostScript FAQ

There is also a POSTSCRIPT-FAQ, consult

```
ftp wilma.cs.brown.edu:  
    pub/comp.lang.postscript.
```

It contains an annotated bibliography as well.

The examples from Adobe's blue book are available on the net.

2.5 Subset 0 from the language

POSTSCRIPT is stack-oriented. This means that operations are prescribed in polish-reverse notation, also known as postfix notation, similar to the HP pocket calculators. Addition—use of operator add—for example is notated as follows.

```
2 3 add%yields 5 on the stack, 2 3 consumed
```

POSTSCRIPT is artificially structured via structure information in comments, double %-ed comments. Programs which obey the Adobe structure are called conforming and this is usually needed for inclusion within (La)TeX, especially the BoundingBox line is required.

Example (Conforming EPS structure)

```
%! PS EPS  
%%Title: <name>  
%%Creator: <name>  
%%CreationData: <date>  
%%BoundingBox: <llx> <ly> <urx> <ury>  
%%DocumentFonts: (atend)  
%%EndComments  
<prolog>  
%%EndProlog  
%%Page: 0 1  
<page 1>  
%%Page: 1 2  
<page 2>  
%%Trailer  
<...>  
%%DocumentFonts: Times-Roman ...  
%%Pages: 3  
%%EOF
```

Creating and drawing paths is done by separate operators. For creating paths operations like `moveto` are provided while drawing goes via `stroke`.

```
0 0 moveto 0 10 lineto%create path  
stroke%draw a v-line of 10pt height
```

Variables—names to be associated with their values—are handled via the so-called dictionaries. The functionality can also be obtained via procedures.

```
/size {10} def
```

The so-called literal name is preceded by a slash to distinguish the declaration from its invoke. The invoke is done

¹¹Nice are the hints to remove repeated parts from files which are generated by CorelDRAW and Mathematica, in order to reduce the size of the automatically generated and to be included files. (The idea is to remove duplicate 'dictionaries' which are included with each result.) The example of how to include graphs of math functions in a document is very useful. However, with respect to his first picture I would prefer to use the inherent symmetry in the data as opposed to providing all the data.

¹²Gurari E.M (1994): *T_EX & L_AT_EX—Drawing & Literate Programming*. McGraw Hill. ISBN 0-07-025208-4.

¹³A complete list with functional summaries is in the red book Section 6.2 Operator summary.

by just the name, also called executable name. The procedure text is surrounded by curly braces. Parameters are absent too. The (operand) stack is used.

For graphics we have a CurrentTransformMatrix—CTM—which maps the user space on the device space, the printer or screen. Equally powerful is the concept of encapsulating graphics via `gsave` and `grestore`, that is the graphics state is local—encapsulated—after `gsave` until `grestore`.

Next to the CTM POSTSCRIPT maintains the `currentpoint` and `currentpath`.

Batagelj, MAPS 95.1—Combining *T_EX* and POSTSCRIPT—provides an in a nutshell overview.¹¹ Another introduction is in Fokker en van Oostrum's 'Plaatjes in een tekst,' MAPS 94.2, next to a survey of drawing software.

2.5.1 Snapshot of (graphics) commands

The following summary is borrowed from Gurari, well ... a little modified.¹² Its main purpose is to show that the number of relevant graphic primitives is low. The functionalities will be dealt with in the examples along the way. For the details of the commands or the list of operators see the red book.¹³

Arithmetic and math operators

```
<num> <num> mul num  
<num> sine num
```

Path construction operators

```
currentpoint x y  
<x> <y> moveto  
<dx> <dy> rmoveto  
<x> <y> lineto  
<dx> <dy> rlineto  
<q1x> <q1y> <q2x> <q2y> <p2x> <p2y> curveto  
<cx> <cy> <r> <ang1> <ang2> arc
```

String operators

```
<string> <num> <num> getinterval
```

Character and font operators

```
/<fontname> findfont  
<fontsize> scalefont setfont  
<string> show  
{<body>} <string> kshow
```

Graphics state operators

```
<num> setgray  
<num> setlinewidth
```

Dictionary operators

```
/<defname> {<body>} def
```

Coordinate system and matrix operators

```
<num> <num> translate
```

$\langle num \rangle \langle num \rangle$ scale
 $\langle num \rangle$ rotate

Relational, boolean, and bitwise operators
 $\langle num_1 \rangle | \langle string_1 \rangle \langle num_2 \rangle | \langle string_2 \rangle$ le bool

Control operators

$\langle bool \rangle \{ \langle truepart \rangle \} \{ \langle falsepart \rangle \}$ ifelse
 $\langle num \rangle \{ \langle body \rangle \}$ repeat
 $\langle from \rangle \langle step \rangle \langle to \rangle \{ \langle body \rangle \}$ for

With postfix notation a sentence like the following is elegant, and close to the familiar input → output.

$\langle in \rangle \langle operator \rangle \langle result \rangle$

2.6 What is not allowed as EPS?

I'm not knowledgeable enough to answer that question, nor do I know of a full-blown definition of EPS.¹⁴ For the moment I consider EPS as some subset which works with all interpreters, with my subset 0 in there. When one restricts oneself to the basics of graphics, arithmetics and similar operations then the boundary between EPS and full POSTSCRIPT—or its various implementations—is not in sight.

2.7 Proofing

For previewing or printing, *as such* I have to include a shift to move the picture away from the lower left corner, say

300 500 translate

2.8 Inclusion

I usually build a figure symmetrically around the origin and then include it in my TeX document via

$\$ \$ \backslash psfig\{file=<name>, height=<number><unit>\} \$ \$$

A unit can be in(ch), cm, and ilks. \psfig is very vulnerable to spaces because of TeX's parsing. So no spaces in there. Now and then I forget to deactivate the translate needed while previewing. No real problem.

2.8.1 BoundingBox

Providing the right BoundingBox coordinates has all to do with proper placement within context, the look-and-feel. Default POSTSCRIPT assumes the origin—in user space—at the lower left corner of the paper—in device space.

Surround the picture by as-if lines and supply the coordinates, in points as units in user space coordinates, of the lower left corner and the upper right corner in the BoundingBox specification. Simple is to build a picture around its symmetry point—and let this coincide with the origin—with as pleasing result that the horizontal positioning comes out centered, when used within math display. Vertically, I add a 10 or so extra on either side in the

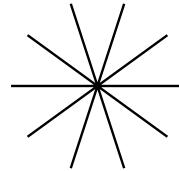
BoundingBox specification, but that depends on the character of the picture.

Some preview systems can measure the BoundingBox and allow adjustment interactively.¹⁵

2.9 Writing PostScript

A line bundle and a variant of it are introduced to show how to create simple EPS.

2.9.1 A line bundle



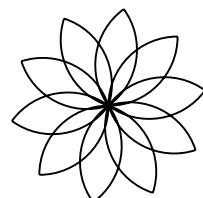
How to do this in POSTSCRIPT? A line as such is simple. First a moveto and then a lineto. So a way is to create a loop and repeatedly draw from the origin to the end of the various lines. This can be done elegantly by using appropriately the CTM.

```
%! PS EPS
%%Title: Line bundle
%%Creator: cgl
%%CreationDate: June 4 1996
%%BoundingBox: -40 -45 40 45
%%Pages: 1
%%EndProlog
%%Page: 1 1
/r 36 def
10{0 0 moveto r 0 lineto
    36 rotate
}repeat stroke showpage
```

Explanation. The idea is that first a simple line is drawn, for example along the x-axes. What happens if after that we rotate? Right, the mapping is changed. And what happens if we supply the *same* line after this? Indeed, it will show up rotated. Because POSTSCRIPT is an interpretive language we can realize this specification after the rotate via a loop, which for this simple case reads 10{...}repeat.¹⁶

Appropriately maintaining the CTM for symmetrical pictures can yield simple looking POSTSCRIPT programs.

2.9.2 A flower



¹⁴Gurari has pointed to some information on the net but it looks informal to me.

¹⁵For a summary of tools to assist finding the BoundingBox coordinates see, Reckdahl K (1995): Using EPS graphics in LATEX documents. reckdahl@leland.stanford.edu or his 1996 TUGboat tutorial.

¹⁶Do you see the variant for drawing a polygon? This duality line bundle and polygon has been used by Gabo and is about what he called stereometry versus perimetry, the structure versus the surface

This exercises the use of arc.

```
%! PS EPS
%%Title: Flower
%%Creator: cgl (Courtesy Papert)
%%CreationDate: June 4 1996
%%BoundingBox: -40 -45 40 45
%%Pages: 1
%%EndProlog
%%Page: 1 1
/r 36 def
10{r r moveto%begin drawing point
  r 0 r 90 180 arc
  currentpoint%origin
  0 r r 270 360 arc
  36 rotate
}repeat stroke showpage
```

Explanation. We have the same structure as the previous program but the ‘line’ is now a little more elaborated: two arcs of a circle. POSTSCRIPT provides an operator for drawing circular arcs, called arc. The arc has (x, y) as centre, r as radius, ang_1 the angle of a vector from (x, y) of length r to the first endpoint of the arc, and ang_2 the angle of a vector from (x, y) of length r to the second endpoint of the arc.¹⁷ These arguments are expected to be on the stack.

```
x y r angl ang2 arc
```

Important is to realize that arc *counts* its angle from $(x, 0)$ and that the *drawing* starts from the point on the stack

The specification of the flower in METAFONT/MetaPost reads essentially as follows.

```
for k:= 1 upto 10:
draw(origin{up}..{right}(up+right){down}...
    {left}origin) rotated 36k;
endfor
```

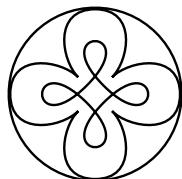
Explanantion. METAFONT allows for specification of the directions¹⁸ $up = (0, 1)$, $right = (1, 0)$.

IMHO, with all respect the METAFONT and POSTSCRIPT programs are similar modulo some syntactic sugar. However, the extra possibility of specifying the directions is more convenient than using control points. But perhaps that is a matter of taste, although the handling of control points is powerful as Bézier himself has shown in the past. From this I conclude that for these simple kinds of pictures we can as well use POSTSCRIPT straightaway.

3 Some more Graphics

Example (Malbork window)

This is all about using curveto, especially choosing suitable control points.



```
%! PS EPS
%%Title: Malbork Window
%%Creator: cgl
%%CreationDate: May 21 1996
%%BoundingBox: -40 -40 40 40
%%Pages: 1
%%EndProlog
%%Page: 1 1
45 rotate 10 0 moveto
4{20 0 37.5 12.5 25 25 curveto
  12.5 37.5 0 20 0 10 curveto
  90 rotate
}repeat%inside loops next
5 0 moveto
4{5 35 35 5 0 5 curveto
  90 rotate
}repeat%enclosing circle next
36 0 moveto
0 0 36 0 360 arc
stroke showpage
```

Explanation. translate changes the CTM, with the effect that the device coordinates are shifted. (Useful for use of POSTSCRIPT alone out of context.)

rotate changes the CTM, and because of being an interpretive language the various loop traversals map the *same* user coordinates on the rotated device coordinates.

$\langle\text{number}\rangle\{\dots\}$ repeat is a loop to be traversed $\langle\text{number}\rangle$ of times.

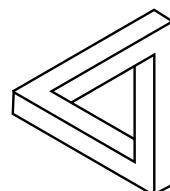
curveto adds a spline to the current path from the currentpoint to the last point on the stack. The first two points are the so-called control points of the spline.¹⁹

arc adds a circular arc to the current path from the currentpoint.

The details of the arguments for the operators are nicely documented in the red book.

Example (Escher’s impossible triangle)

This is all about *wrong* projections. However, these kinds of pictures are intriguing and fun. I consider them well-suited to illustrate POSTSCRIPT’s drawing capabilities.



```
%! PS EPS
%%Title: Escher's impossible triangle
%%Creator: cgl (inspired by Guy Shaw)
%%CreationDate: May 23 1996
%%BoundingBox: -40 -40 40 40
%%Pages: 1
```

¹⁷The arc is drawn counter clockwise. arcn draws clockwise.

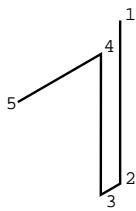
¹⁸There is also a quartercircle which apart from orientation is drawn similarly.

¹⁹Much similar as in METAFONT. Choosing for the inner lop the control points in this way is borrowed from Haralambous Y (1995): Some METAFONT techniques. TUGboat 16.1, 46–53. It is also supplied in the description of curveto in the red book.

```
%%EndProlog
%%Page: 1 1
3{25 34 moveto
 25 -34 lineto
 17 -38.2 lineto
 17 20 lineto
-17.6 0 lineto
120 rotate
}repeat stroke showpage
```

5 points, the right stroke and a rotation or two, that's it.
End of story.

However, it is all about finding those 5 points.

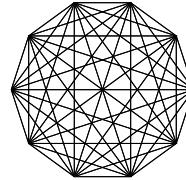


```
-120 rotate
point lineto%preserve symmetry
120 rotate
currentpoint 2 div neg lineto
currentpoint 3 sqrt mul sub 0 lineto
120 rotate
}repeat stroke showpage
```

Explanation. `currentpoint` yields the coordinates of the current point of the path on the stack. The other operations do what their names suggest. The temporarily change of the CTM within the loop expresses the rotation symmetry relation between points 1 and 3.

Example (Bentley's polygon)

This code is all about a double loop and using the loop variable from the stack, next to using the `gsave` and `grestore` advantageously.



```
! PS EPS
%%Title: Essential stroke
%%Creator: cgl (inspired by Guy Shaw)
%%CreationDate: May 23 1996
%%BoundingBox: -40 -40 40 40
%%Pages: 1
%%EndProlog
%%Page: 0 1
25 34 moveto currentpoint
0 -68 rlineto currentpoint%down
-120 rotate
25 34 lineto%preserve symmetry
120 rotate currentpoint
17 20 lineto currentpoint
-17.6 0 lineto currentpoint
%labels
/Courier findfont 8 scalefont setfont
moveto -5 -3 rmoveto (5) show
moveto 1 1 rmoveto (4) show
moveto 2 -5 rmoveto (3) show
moveto 2 0 rmoveto (2) show
moveto 2 0 rmoveto (1) show
stroke showpage
```

Explanation. The essential stroke figure also illustrates the integration of text in this case digits. `currentpoint` pushes the point on the stack. The last `moveto`'s pop these coordinates up. `rmoveto` moves *relatively*.

And what about their relationships, and what about the minimal information to be prescribed?

Looking more closely it turns out that *only* the first point is all that is needed. The rest is implicit to the nature of the figure.²⁰

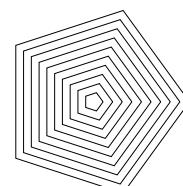
```
! PS EPS
%%Title: Escher's Impossible triangle II
%%Creator: cgl
%%CreationDate: May 23 1996
%%BoundingBox: -40 -40 40 40
%%Pages: 1
%%EndProlog
%%Page: 1 1
%Parameterized over p1
/point {25 34} def%note x<y
%
3{point moveto
currentpoint neg lineto%down
```

```
! PS EPS
%%Title: Bentley's double loop
%%Creator: cgl
%%CreationDate: May 30 1996
%%BoundingBox: -100 -105 100 105
%%EndProlog
10{1 1 9{100 0 moveto
  gsave
  36 mul rotate%loopcount*36
  100 0 lineto stroke
  grestore
} for
36 rotate
}repeat showpage
```

Explanation. `gsave` and `grestore` are needed to draw locally, that is at the end the graphics state—`currentpoint`, `currentpath` and `CTM`—is restored with the values at the beginning. 1 1 9 stand for beginvalue step and endvalue of the `for` counter.

Example (Another double loop)

A set of nested polygons provide also a double loop situation.



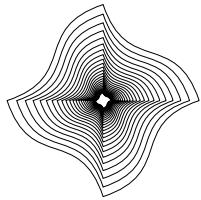
```
! PS EPS Nested pentagons
%%Title: Pentagons
%%Creator: cgl
%%CreationDate: June 17 1996
%%BoundingBox: -100 -100 100 100
%%Pages: 1
```

²⁰Of course one can also think of other equivalent parameters like size and thickness.

```
%%EndProlog
%%Page: 1 1
10 10 100{dup 0 moveto
  5{72 rotate
    dup 0 lineto
  }repeat
}for stroke showpage
```

Example (Polygons with splines as sides)

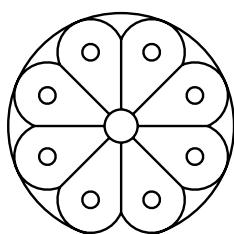
This generalization of polygons was introduced by Jackowski at EuroTEX 95. A special case of METAFONT's interpath functionality is shown en-passant.



```
! PS EPS Nested 'squares'
%%Title: polygon.eps II
%%Creator: cgl
%%CreationDate: June 17 1996
%%BoundingBox: -100 -100 100 100
%%Pages: 1
%%EndProlog
%%Page: 1 1
/r 100 def
/rl1 {r .25 mul} def
/r3 {r .75 mul} def
25{r 0 moveto
  4{r3 r3 rl1 rl1 0 r curveto
    90 rotate
  }repeat
  .9 .9 scale
}repeat stroke showpage
```

Example (Barn window)

This is all about playing with circles and circular arcs.²¹

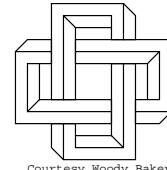


```
! PS EPS
%%Title: Barn Window II
%%Creator: cgl
%%CreationDate: May 29 1996
%%BoundingBox: -45 -45 45 45
%%Pages: 1
%%EndProlog
%%Page: 1 1
/l 36 def
/r {l 22.5 sin mul} def
/m {l 22.5 cos mul} def
8{r .5 mul 0 moveto
  l 0 lineto
  currentpoint %begin circular arc
  22.5 rotate m 0%center
  r %radius
  -90 90 arc
  22.5 rotate
}repeat
```

```
%inner circle
/rin {r .5 mul} def
rin 0 moveto
0 0 rin 0 360 arc
%outer circle
/rout {r m add} def
rout 0 moveto
0 0 rout 0 360 arc
%extra circles
/rin {r .25 mul} def
22.5 rotate
8{m rin add 0 moveto
  m 0 rin 0 360 arc
  45 rotate
}repeat stroke showpage
```

I'm sure I'll come back some day and look again through this window, but then pastel colored.

Example (Baker's inspiration)



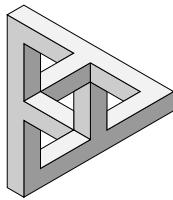
Courtesy Woody Baker

This example is similar to Escher's impossible triangle. Find the essential stroke and rotate.

```
! PS EPS
%%Creator: cgl (inspired by Woody Baker)
%%CreationDate: May 1996
%%BoundingBox: -80 -80 80 80
%%Pages: 1
%%EndProlog
%%Page: 1 1
4{-15 25 moveto
  0 -10 rlineto
  60 0 rlineto
  0 -30 rlineto
  10 0 rlineto
  0 40 rlineto
  -70 0 rlineto
  0 10 rlineto
  80 0 rlineto
  0 -60 rlineto
  -30 0 rlineto
  0 10 rlineto
  10 0 rlineto
%
  35 -25 moveto
  0 -10 rlineto
  20 0 rlineto
  10 10 rlineto
%
  45 15 moveto
  10 10 rlineto
90 rotate
}repeat stroke
%
/Courier findfont 10 scalefont setfont
-55 -75 moveto
(Courtesy Woody Baker)show
showpage
```

²¹The first example in the blue book collection provides a similar picture with gradually changing scales of grey.

Example (Romanovsky's real Escher)

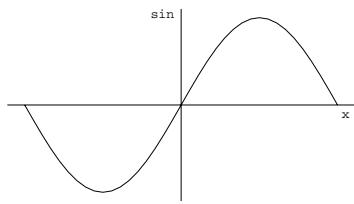


Grey scales can be obtained simply via `<number>` `setgray`, with $\langle number \rangle \in [0, 1]$. 0 denotes black and 1 is white. The idea is to construct the essential path—the stroke denoted by a grey scale—and to use this 3 times.

4 Math graphs

In (La)TeX documents it is a problem²² how to include accurate graphs of mathematical functions. Because of POSTSCRIPT's arithmetic and graphics capabilities it is handy to use POSTSCRIPT.

Example (Sine function)



```

%! PS EPS
%%Title: Sine function
%%Creator: cgl (inspired by Batagelj)
%%CreationDate: May 27 1996
%%BoundingBox: -200 -110 200 110
%%EndProlog
/Courier findfont 15 scalefont setfont
% $x$ -axes and label
-200 0 moveto 200 0 lineto
-15 -15 rmoveto (x) show
% $y$ -axes and label
0 -110 moveto 0 110 lineto
-35 -10 rmoveto (sin) show
%function
-180 0 moveto
-180 10 180{ %from step to
    dup sin 100 mul%(x, 100sin x)
    lineto
}for stroke showpage

```

The invoke might read as follows.

```
 $$\backslash psfig\{file=sine.eps, height=1in\}$$
```

5 Text set along curved paths

A teaser. With the advent of scalable and rotationable outline fonts this is possible too.²³

Example (Along a circle)

h i r t q u e

```
! PS EPS
%%Title: Typesetting along arcs
%%Creator: cgl
%%CreationDate: June 4 1996
%%BoundingBox: -100 50 100 125
%%Pages: 1
%%EndProlog
%%Page: 1 1
/Courier findfont 10 scalefont setfont
/text (happybirthday) def
50 rotate
0 1 12{0 100 moveto
        text exch 1 getinterval show
        -10 rotate
}for stroke showpage
```

Joseph Romanovsky communicated that kshow— kerning (and more general positioning) under user control—is available which allows a general def to be executed between two characters of a string.

Example (Along a spiral)

The blue book provides an example of typesetting along a path—a quotation of Woody Allen—where the path accentuates his filmmaker profession. The example below shows a nice effect with little knowledge of POSTSCRIPT, essentially the use of `kshow`.

Olga Grineva
My charming
nun
Peterburg hostess

```
%! PS EPS
%Title: Text along spiral
%Creator: J.V. Romanovsky
%CreationDate: Adapted from JVR June 96
%BoundingBox: -100 -90 60 70
%EndProlog
/Courier findfont 20 scalefont setfont
-100 0 moveto 50 rotate
{-10 rotate 3 0 rmoveto .98 .98 scale}
(Olga Grineva my charming
 St Peterburg hostess)kshow
showpage
```

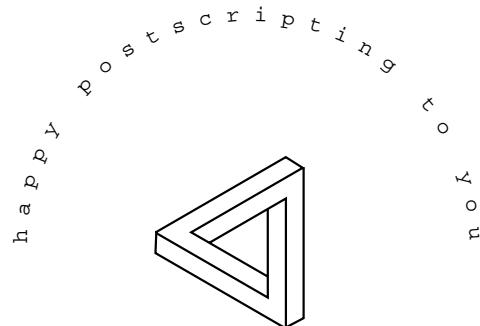
²²Communicated by Nico Temme. He solved the problem by doing the calculations in PASCAL. For advanced manipulations Mathematica or Mapple are generally used where the resulting EPS is pasted up in the (La)TeX script as usual.

²³Disclaimer: Typesetting math along curved paths is something different.

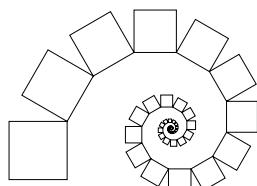
Example (Seals)

The problem has been discussed by Hoenig at EuroTeX 92, and Zlatuška at EuroTeX 95, both biased by METAFONT. POSTSCRIPT alone is suited too with an overall simpler process. Combining two earlier supplied examples yields Zlatuška's seal in principle.²⁴

```
%! PS EPS
%%Title: Seal, in principle
%%Creator: cgl
%%CreationDate: June 6 1996
%%BoundingBox: -110 -45 110 100
%%Pages: 1
%%EndProlog
%%Page: 1 1
%150 650 translate
/Courier findfont 10 scalefont setfont
/text (happy postscripting to you) def
/r 100 def
gsave
 90 rotate %begin orientation
 0 r moveto%begin point
{-7.04 rotate 0 r moveto} text kshow
grestore%next the central Escher
3{25 34 moveto
 25 -34 lineto
 17 -38.2 lineto
 17 20 lineto
-17.6 0 lineto
120 rotate
}repeat stroke showpage
```



Remark. The difference of fonts in the main text and that used by POSTSCRIPT is no longer there when POSTSCRIPT fonts are used throughout, be it the POSTSCRIPT version of the CM family.

Example (Gurari's squares)

```
%! PS Gurari squares
%%BoundingBox: -200 -110 200 110
%%Creator: cgl
%%CreationDate: June 20 1996
%%EndProlog
```

```
/r 22 def
/square {1 1 4{0 r rlineto
 90 rotate}for} def
0 0 moveto 90 rotate
50{r 0 rmoveto square
 -30 rotate .9 .9 scale
}repeat stroke showpage
```

Example (Gurari's ABC)

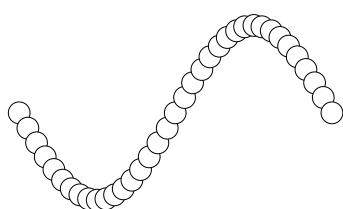
Very nice this suggestion of motion.



```
%!PS EPS
%%Title: Gurari's ABC
%%BoundingBox: 0 -45 100 75
%%Creator: Gurari
%%CreationDate: copied June 17 1996
%%Pages: 1
%%EndComments
%%EndProlog
%%Page: 1 1
/Times-Bold findfont 45 scalefont setfont
-40 rotate
1 -.03 0{setgray
 0 0 moveto
 (ABC) show
 3 rotate
 } for
0 0 moveto -4 rotate
1 setgray (ABC) show
showpage
```

Example (Walking along the S-curve)

In *The METAFONTbook* ex13.10 is about drawing overlapping disks along a path, the S-figure. How to do this in POSTSCRIPT straightaway? There is no 'point of' a path operator so the best we can attain is to walk along a math function.²⁵



```
%!PS Sine with overlapping disks
%%BoundingBox: -200 -110 200 110
%%Creator: cgl (METAFONTbook ex13.10)
%%CreationDate: June 17 1996
%%EndProlog
newpath
-180 10 180{dup sin 100 mul%(x, 100sin x)}
```

²⁴The blue book also provides a seal—*Symphony No.9*—but that is more complex and ipso facto requires more knowledge of POSTSCRIPT to understand what is going on, IMHO, with all respect. To set this poster from the blue book is no more difficult than the use of *arc*, however. My example gives you the feeling that you understand what is going on.

²⁵Or specify a path explicitly of course.

```

12.5 0 360 arc
gsave 1 setgray fill grestore
stroke
}for showpage

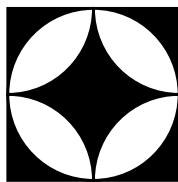
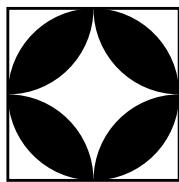
```

Explanation. 1 `setgray fill` is the erase functionality, encapsulated to yield what we want.

6 Reverse video

Let us go back to the flower picture as given at the beginning of this note and reuse the leaves.

Example (reverse video)



The above is obtained via

```

%\PS Reverse video
%%BoundingBox: -30 -35 120 35
%%Creator: cgl
%%CreationDate: Aug 1996
%%Pages: 1
%%EndProlog
%%Page: 1 1
/r 30 def
/filling {fill} def
/tile {4{r 0 moveto
  0 0 r 0 90 arc
  currentpoint
  r r r 180 270 arc
  filling
  90 rotate
  }repeat
} def
/frame {r neg r moveto
r 2 mul 0 rlineto
0 r -2 mul rlineto
r -2 mul 0 rlineto
closepath
} def
%
%tile
%
gsave
  tile frame stroke
grestore
%
%reverse video tile
%
r 3 mul 0 translate
frame
gsave fill grestore%background
stroke
/filling{gsave 1 setgray fill
  grestore}def
tile stroke showpage

```

Reverse video in POSTSCRIPT can be obtained via providing a black background and for the picture replace `fill` by `gsave 1 setgray fill grestore`.

7 Tiling

Tiling is all about copies of an element, shifted and/or rotated, to fill up space traditionally in the plane. Below a

leave is ‘copied’ four times and the resulting tile is ‘copied’ four times. The copying comes down to redoing the figure at the prescribed place eventually rotated. The latter is possible by modifying the CTM via `translate` or `rotate`.

Example (Tiling)



The above is obtained as follows.

```

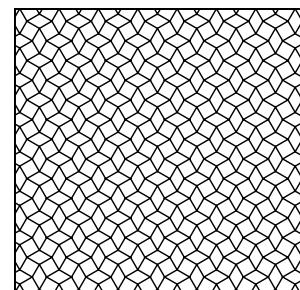
%\PS Tiling and reverse video
%%BoundingBox: -45 -40 40 45
%%Creator: cgl
%%CreationDate: Aug 1996
%%Pages: 1
%%EndProlog
%%Page: 1 1
/r 20 def
/tile {4{r 0 moveto
  0 0 r 0 90 arc
  currentpoint
  r r r 180 270 arc
  fill
  90 rotate
  }repeat
} def
/frame {r neg r moveto
r 4 mul 0 rlineto
0 r -4 mul rlineto
r -4 mul 0 rlineto
closepath
} def
%
frame stroke%no clipping necessary
2{2{tile r 2 mul 0 translate
  }repeat r -4 mul r -2 mul translate
}repeat
r r neg translate
showpage

```

8 Clipping

The clipping functionality is different in spirit from METAFONT. In POSTSCRIPT we have to adjust the frame to draw within via creating a path and clip this path, that is make this path the drawing boundary, that is all.

Example (Clipping)



The above is obtained via

```

%\PS Tiling fourthree
%%BoundingBox: -100 -100 100 100
%%Creator: cgl

```

```

%%CreationDate: Aug 1996
%%Pages: 1
%%EndProlog
%%Page: 1 1
%300 500 translate
/a 10 def
/ha {a .5 mul} def
/tile {%
    rhombus + 90 rotated rhombus
    ha 3 sqrt mul 0 moveto
    0 ha lineto
    ha 3 sqrt mul neg 0 lineto
    0 ha neg lineto
    closepath
    ha 1 3 sqrt add mul ha 3 sqrt mul lineto
    ha 2 3 sqrt add mul 0 lineto
    ha 1 3 sqrt add mul ha 3 sqrt mul neg lineto
    closepath
} def
/tena {a 10 mul}def
/frame {tena neg tena moveto
tena 2 mul 0 rlineto
0 tena -2 mul rlineto
tena -2 mul 0 rlineto
closepath
} def
/dotiling {
    a -11 mul tena neg translate
    9{gsave
        11{tile a 1 3 sqrt add mul 0 translate
            }repeat stroke
        grestore
        gsave ha 1 3 sqrt add mul dup translate
        11{tile a 1 3 sqrt add mul 0 translate
            }repeat stroke grestore
        0 a 1 3 sqrt add mul translate
    }repeat
} def
%
%tile
%
frame clip dotiling showpage

```

9 Tables set sideways

Another teaser is to set tables rotated. Rokicki provided rotate macros along with his dvips, among others. I have borrowed the essence from his rotate macros and recast into the following.

```

\def\rotate#1%stuff
    #2%degrees in PS direction
{\setbox\abox=\hbox{\#1}%
 \adim\ht\abox\advance\adim by\dp\abox
 \hbox to\adim{\vbox to\wd\abox
 {\vskip\wd\abox
 \special{ps: gsave
     currentpoint currentpoint translate
     #2 neg rotate
     neg exch neg exch translate}%
 \box\abox\vss}\hss}%
 \special{ps: currentpoint
         grestore moveto}%
}%end rotate

```

The point is that it is not much. The `ps:` is dependent on the system still, alas.

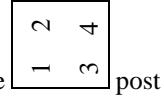
Example (Rotated table)

The example works under UNIX with Rokicky's dvips.

```

\def\data{1\cs2\rs
          3\cs4 }
pre
\rotate{\framed
        \btable\data} pre
        {90}
post

```



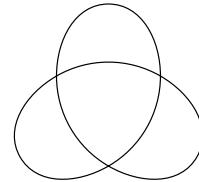
Remark. Gurari (1994) has provided some more examples of POSTSCRIPT↔(AI)DraTeX interaction. Real interfacing. Apart from portability problems it gets quite complicated. For the moment I refrain and code the ‘pictures’ in raw POSTSCRIPT assisted by METAFONT for prompting control points of those curves which can be specified elegantly in a declarative way. And, of course, there is the wealth of PStricks.

10 METAFONT/MetaPost user interface

Sometimes it is more natural to specify points and *directions*. It is true that specifying a control point along the direction can yield the same effect but the distance between the point and its control point influences the shape.

Example (Escher’s knot)

This example is all about specifying directions.



In METAFONT the coding would read as follows.

```

%Escher's Knot. June 96. cgl@rc.service.rug.nl
def openit = openwindow currentwindow
    from origin to (screen_rows,screen_cols)
    at (-2r,3r)enddef;
pickup pencircle scaled 1;
tracingstats:=proofing:=1; screenstrokes;
pair p[];
%parameters
r:=100; alfa=90;
%
p2:=(0,.85r); %independent from p1,3,4
p4:=(0,-.5r);
%dependent points because of symmetry
p1:=p4 rotated -120;
p3:=p4 rotated 120;
path q;
q=p1{dir alfa}..{(1,0)}p2..
    {dir(-alfa)}p3..{dir(alfa-240)}p4;
draw q;
draw q rotated 120;
draw q rotated-120;
showit;
end

```

By the nature of the figure not only points are related but also their directions. How to cope with this in POSTSCRIPT? It can be done but not so elegantly, honestly speaking it is quite cumbersome. But ...there is a solution or two, hang on.

```
%! PS EPS
```

```

%%Title: Escher knot II
%%Creator: cgl (inspired by Knotplot)
%%CreationDate: June 1996
%%BoundingBox: -95 -95 95 95
%%Pages: 1
%%EndProlog
%%Page: 1 1
%
/angle 90 def
/r 100 def
/point {0 -.5 r mul}def
/p1 {-25 r mul 3 sqrt mul .25 r mul moveto
    currentpoint
    angle sin 2 mul add exch
    angle cos 2 mul add exch
    -20 .85 r mul
    0 .85 r mul
    curveto stroke} def
/p3 {.25 r mul 3 sqrt mul .25 r mul moveto
    %Control point
    currentpoint
    angle sin -15 mul add exch
    angle cos 15 mul add exch
    %Control point:
    % 58.62 -.5 r mul 5 add%angle 90
    0 angle -240 add cos -15 mul add
    -.5 r mul angle -240 add sin -15 mul add
    0 -.5 r mul
    curveto stroke} def
3{p1
  gsave -1 1 scale p1 grestore%reflect
  p3
  120 rotate
}repeat showpage

```

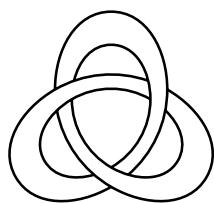
Explanation. The essential curve is split into 3 pieces: p_1, \dots, p_3 . The first two are related by reflection. The third must properly match. In the Columbus's egg paragraph the straight .eps code is given, biased by the knowledge of the (control) points.

10.1 A teaser

More complicated is when the line is changed into a tube, and when we have to deal with hidden lines. In METAFONT the code could read as follows, where use is made of `intersectiontimes` and of subpaths.²⁶

Example (Escher's doughnut)

This example is all about getting from a *declarative* specification in METAFONT to an *imperative* EPS code.



First the declarative METAFONT code.

```

%Escher Knot III. June 96.
% cgl@rc.service.rug.nl
def openit = openwindow currentwindow

```

²⁶This code works as such on my Mac with Bluesky's PD METAFONT. For other environments build a character from it, or adapt it for use in MetaPost, or even simpler copy the bread-and-butter EPS code which is appended at the end.

²⁷A white lie. I have edited the file and reduced the data—and deleted `dtransform`, `idtransform` and the various `set...—` for the 6 strokes into only 2 and rotated these. METAFONT allowed me to declaratively specify the picture while MetaPost provided me with the essential path data. Well...even METAFONT can be asked to provide those (control) points.

```

from origin to (screen_rows,screen_cols)
at (-2r,3r)enddef;
pickup pencircle scaled 1;
tracingstats:=proofing:=1; screenstrokes;
numeric t, u, v, w;
pair p[]; path q[];
def assignpoints=
  p2:=(0,.85r); %independent from p1,3,4
  p4:=(0,-.5r);
  %dependent points because of symmetry
  p1:=p4 rotated -120;
  p3:=p4 rotated 120;
enddef;
%
alfa=90;r:=100; assignpoints;
q1:=p1{dir alfa}..{(1,0)}p2..
  {dir(-alfa)}p3..{dir(alfa-240)}p4;
%inside
  r:=.75r; assignpoints;
q2:=p1{dir alfa}..{(1,0)}p2..
  {dir(-alfa)}p3..{dir(alfa-240)}p4;
(t,u)= subpath (2.5,3) of q1
  intersectiontimes (q2 rotated -120);
(v,w)= q2 intersectiontimes
  (q1 rotated 120);
%showvariable t,u;
draw subpath (0,2.5 + t/2) of q1;
draw subpath (0,2.5 + t/2) of q1
  rotated 120;
draw subpath (0,2.5 + t/2) of q1
  rotated-120;
%showvariable v,w;
draw subpath (v,3) of q2;
draw subpath (v,3) of q2 rotated 120;
draw subpath (v,3) of q2 rotated-120;
showit;
end

```

To give the reader an impression of what MetaPost will yield `escherknotIII.eps`, the imperative code, is included.²⁷

```

%! PS EPS
%%BoundingBox: -40 -31 40 43
%%Creator: MetaPost and JJW, cgl
%%CreationDate: June 17 1996
%%Pages: 1
%%EndProlog
%%Page: 1 1
3{-21.6507 12.5 moveto
-21.6507 27.74551 -13.78212 42.5003
  0 42.5003 curveto
13.78212 42.5003 21.6507 27.74551
  21.6507 12.5 curveto
21.6507 -0.24506 16.04897 -12.19249
  6.58395 -20.3313 curveto
%
-14.3152 15.86746 moveto
-12.43301 23.58928 -7.45113 29.75021
  0 29.75021 curveto
9.64748 29.75021 15.15549 19.42186
  15.15549 8.75 curveto
15.15549 -2.07904 9.37823 -12.08548
  0 -17.5 curveto
120 rotate
}repeat stroke showpage

```

As can be seen from the last code it is all about finding the right (control) points and draw the strokes, as remarked at the beginning of this note. The difference between the declarative METAFONT specification and the

resulting (unedited) MetaPost code is striking. When the last code is shown first one would say, ah... POSTSCRIPT is easy just data and some strokes. The resulting code is equivalent to Woody Baker's code: just the right stroke and a rotation or two.

KnotPlot on the net provides a more complicated version where the light reflection is emulated by shades of grey. The gzipped file is 64KB, however. A world of difference.

10.2 Columbus' egg

Why not use METAFONT to create 'the (control) points' from the descriptive picture and use these in raw POSTSCRIPT straightforward?

After assigning `precontrol-s` and `postcontrol-s` to pairs and inserting `show-s`, METAFONT yielded for the simple Escher knot the data in the transcript file. A little editing of this log file resulted in the following imperative EPS code.

```
%! PS EPS
%%Title: Escher knot (mf prompted)
%%Creator: cgl
%%CreationDate: June 1996
%%BoundingBox: -80 -80 80 80
%%Pages: 1
%%EndProlog
%%Page: 1 1
3{-43.30139 25 moveto
-43.30139 55.49103 -27.56424 85.00061
0 85.00061 curveto
27.56424 85.00061 43.30139 55.49103
43.30139 25 curveto
43.30139 -5.94014 26.79497 -34.5299
0 -50 curveto
120 rotate
}repeat stroke showpage
```

I presume the functionality is similar to Jackowski's `mftoeps`. The above method is my Poor Man's METAFONT2EPS, with concise, very concise and intelligible EPS as result.²⁸

11 Acknowledgements

First of all Don Knuth and John Hobby thank you.

Thank you Joseph Romanovsky for showing by example the power of POSTSCRIPT, and for your cooperation on the METAFONT↔POSTSCRIPT duality.

Thank you Bogusław Jackowski for your 'POSTSCRIPT for \TeX ies' at Bacho \TeX 96, suggesting that POSTSCRIPT as such is beneficial for \TeX ies, next to your `mftoeps`.

Thank you Eitan Gurari and anonymous \TeX ies from whom I borrowed material, not in the least Adobe for providing POSTSCRIPT to start with.

Piet Tutelaers provided me with a copy of the PSFAQ, and Erik Frambach traced the file with examples from the blue book, next to KnotPlot.

As usual Jos Winnink proofed the paper and lend a helping hand in procrusting towards MAPS inclusion if not for

processing `escherknot.mf` into `escherknot.eps` via MetaPost, although I could have done without as discussed.

Finally, thank you Erik and Wietse for the various discussions about \TeX and METAFONT.

12 Conclusion

To code symmetrical and simple curves in raw POSTSCRIPT is fun and yields elegant scripts and concise files. To merge text with graphics is fun too, and the teaser to set along curved paths can be done by POSTSCRIPT elegantly. Another teaser of drawing math curves accurately along with (La) \TeX is solved also by means of POSTSCRIPT. Powerful too is to *extend* the inclusion of .eps files at the dvi level by a little more interaction between (La) \TeX and POSTSCRIPT. Rotating a box, with as applications for example typesetting tables in landscape, is possible in POSTSCRIPT, at the expense of system dependency because of the \special-s.

Merging a little knowledge of POSTSCRIPT with \TeX pertise is powerful. PStricks concentrates on *interfacing* (La) \TeX with POSTSCRIPT at the expense of burdening (La) \TeX too much, IMHO, with all respect. For \rotate I interfaced too. However, for typesetting along curved paths I would not think of interfacing via rotated boxes or so.

Of course, people who do need advanced features or have special wishes might better use Adobe Illustrator, CorelDRAW, Mathematica, or Adobe Photoshop, and not to forget the pleasing MetaPost.

To understand and learn \TeX did take me a couple of years. To acquaint myself with METAFONT did cost me a few months. Learning just a little bit of POSTSCRIPT was a matter of weeks, and when concentrating on paths and (control) points the `moveto`, `arc` and `curveto` can be grasped on a late afternoon.

Although the blue book contains also examples of typesetting text, I consider \TeX unsurpassed for this. The best of both worlds is to combine (La) \TeX and POSTSCRIPT.

Maybe we should follow Adobe and extend the use of POSTSCRIPT by PDF—or use the alternative HTML—to facilitate WWW surfing.

13 What more?

For pictures I use a \TeX controlled database with the beneficial side-effect that I don't have to worry about file systems when using pictures (tools, references and ilks) on different machines. I would welcome a similar functionality for my collection of POSTSCRIPT pictures to be used with \psfig. I hope that the examples of the blue book next to my examples as included here—and those to

²⁸Note that the final digits are 'noise.'

come—will contribute to the .eps library²⁹ for reuse or inspiration.

My anthology of examples in METAFONT will emerge in a series of notes with occasionally POSTSCRIPT alternative (hand) codings added. The first note in the series is about tiling.

A next step is the manipulation of colors either via MetaPost or POSTSCRIPT directly. Jackowski uses Adobe Illustrator for example to enrich interactively the systematic EPS pictures created by METAFONT. Indeed interesting, very interesting, but beyond my possibilities for some time to come. Neither do I have access to color POSTSCRIPT printers as yet, alas. My case rests.

Have fun, and all the best.

²⁹This library was coined by Jackowski and Ryćko at EuroTeX 94 to start with their ‘expanded stroke’, ‘removing overlap,’ and ‘updateB(ounding)B(ox)’ if not for their mf2eps package.