A MACRO ROUTINE FOR WRITING TEXT ALONG A PATH IN METAPOST.

SANTIAGO MUELAS

ABSTRACT. In this article we show a general macro written in *pure metapost* for putting *any* text using *any* font over *any* path. The routine will be explained in detail and some graphics will be included for clarifying purposes.

1. INTRODUCTION

In one of the steps of the construction of the program METAGRAF, we feel ourselves obliged to find a method for writing curved text. —Our goal with METAGRAF is to give TEX/LATEX users a simple and strong tool for graphics inclusion in text pages.— We knew the very simple way to achieve this using PsTricks, and at a certain moment we had the temptation to change to this language. Thanks to the very valuable advice of some important names of the *Meta World*, we decided to do it the *hard way* and we stopped the development of our program to concentrate in writing a routine general enough to be included as a general macro in MetaPost packages.
After one month of hard work to find something that could fulfill our *desiderata* we think that finally we have it. This *something* in the form of a quite simple metapost routine is what will be shown and explained in this paper.

2. GLYPH AND BOXES

Being our knowledge of Metapost limited —as we think it is for the biggest part of its users— the first task to achieve was to find the way to measure characters or, more precisely, glyph and their corresponding boxes. As Metapost is a sibling of Metafont and knowing the very systematic way used by D. Knuth in his studies and great creations, we were sure that some way could be found because, in fact, Metapost uses TEX for translating math text.

We looked for a hint to our problem in Hobby's User Manual, but although a last page in it shows a way to recover separate parts of a picture, this was not a big help for our problem, and the treatment of text in this Manual is extremely sober. Nevertheless, two well known capabilities of Metapost were going to be the foundation for the construction of our routine:

- The capability of finding the size of the bounding box of a picture.
- The capability of obtaining a substring from a string.

Date: September 10, 2000.
Key words and phrases. TXP, MetaPost, LATEX, METAGRAF, awk, TEX.
Special thanks are due to Juan J.Arribas, Hans Hagen and Boguslaw Jackowsky.

After a few days employed in looking for some *miraculous way* of achieving our goal, it was clear that the good way was already found. The only real need was to assure that the bounding boxes had no margins —truecorners:=1 was o.k.— and that it was possible to obtain the *length* of a string —in number of characters— as in the Manual this point is unclear. After looking at a few routines in the distribution and having found a couple in which this *length* is obtained, things became more and more clear.

3. The big strength of MetaPost

So, the first part of our job was almost finished. We only needed to write and test it. This first part was to find the glyph one by one and the size of their corresponding boxes. Once done, we will know how to *rewrite* the string with the glyph in the exact place, regarding their separation —positioning them in due vertical point was the third and last part. We will speak about the second part in a few lines.

Now, lets write the few lines needed to accomplish our first part. The steps will be:

(1) Convert the string to a picture and read the total length.
(2) Change the string to another without the last character.
(3) Repeat the first step. The difference between both lengths will be the width of this last character.
(4) Consider that the original string is the second obtained and repeat all the steps. We will obtain the width of the last glyph before the end.

We see that this is very simple to program as it is a loop. The lines of code will be:

```
--------------------------
%    input s & pi
     truecorners:=1; string s,ps; path pi; picture pic, pt;
     for i = length s step -1 until 1:
       ps:=substring(0,i) of s;
       pic:=thelabel(ps, (0,0));
       long[i] = 2*(xpart urcorner pic);
     endfor;
--------------------------
```

After this few lines, we will have the distance from the origin of all our glyph stored in the array **long**[]. It is necessary a second loop to put every single one in due place in the path[1]:

```
--------------------------
     for i = 0 upto (length s) -1:
       sp:= substring(i,i+1) of s;
       if sp <> ‘‘ ’’:
         x:= (long[i] + long[i+1])/2;
         pt:= thelabel(sp, (0,0)) shifted x;
```

[1]We will speak something more about this second loop later on.

```
%     Here the instruction to draw the glyph <---------
        fi;
      endfor;
--------------------------
```

Now, we need to solve the second part of our problem. Our glyph and their separation is known. We could put them in any place, but if we want to put them in a certain line —the path— we need to know how to do that. We need to know the length along the curve to be used and the angle that the curve forms with the coordinates axis in the different points of placement. It seems quite a hard task, but we must not forget that now we are dealing with one of the strongest points of Metapost and all that is needed and already mentioned can be written in just one line of code[2]. —Metapost's math capabilities can be quite impresives.— This line is the one that correspond to our second loop where we have signaled it and must be included there. Here is the line:

```
--------------------------
    draw pt shifted (-xpart pt, h) rotated angle direction arctime x
    of pi  of pi shifted point arctime xpart pt of pi  of pi;
--------------------------
```

It must be said now that the variable **h** that appears in this code, has not been yet obtained. This will correspond to the third part of our search and represent the *height* to place the glyph.

4. Working with fonts

The moment has arrived in which we have to enter in the third part of our *business*. The one to obtain the detailed knowledge of every glyph. And for this part we must say that there is little to be done with Metapost, being the discussion centered in a matter more generic than what we have already seen[3].

In this part, our goal was to know as exactly as possible, the size of the glyph's boxes to be used in the string to put along a curve. Not only the sizes of the boxes but also the situation of the baseline in these boxes. After a couple of days looking for a way to find this data without the need of knowing the specific font used, we have realized that this is not possible, so we have gone directly to the sources, that is: the study of fonts. As it does not seem to be a matter closely related with Metapost, we will not explain in detail the steps done, but we will go directly to the results obtained and the way the final data are used by TXP. There are some points that will be useful to note, and we will do.

- There exists two type of files related with fonts and usually we can find them in any LaTeXdistribution. Those are the files related with the geometry of the boxes and the position of the baseline in them and the files where the shape of the glyph are established. We are concerned with the first ones.
- The files where all the description needed usually resides are the ones with the extension *.afm. Unfortunately, this files are not updated and do not

[2]This line of code has been written by Juan J. Arribas

[3]For this part of our work we have found an enormous help in the book "TEXUNBOUND" from *Alan Hoenig*.

correspond with the precision needed to the real fonts used. The results obtained were just not perfect.

- Another group of files that can be used are those with extension *.pl. They contain a lot of information not needed but they have an enormous advantage over the metrics —or *.afm— ones. For understanding the advantage, lets mention another group of files.
- The files needed by the computer to be able to use a font, are those with the extension *.tfm[4]. Those files are not readable. Nevertheless, usually there is an application found on every distribution of LaTeXcalled **tftopl** that creates a file *.pl from a file *.tfm.
- Now we can understand the advantage mentioned. Once located the files with extension *.tfm that are used by the computer, we can obtain the corresponding *.pl files, that are readable and detailed. Everything needed by our macro regarding fonts is included in the *.pl files. And we can obtain these files from the *.tfm actually used for writing. That means that we can obtain an **absolute precision** with this system. So, this is the system adopted by TXP.

The need to create the files is something to be done once for every font. When they exist, we don't need to recreate anymore...until the changes of the original fonts.
In those files TXP will find an array that relates the number of the ASCII character with the height to place the center of the box of the glyph over the baseline. Something like: hig[65]:=3.2576, for the letter **A**. We will explain later on how to create those files. For the shake of understandability, let's suposse that we have already obtained the arrays.

5. Putting everything in Place

We can come back to our routine to arrange all our knowledge and write it in its totality —for the moment. In the second loop, the one that will put the glyph in due place, we let a gap and write in it:*Here the instructions to draw the glyph.*
The method used by TXP is to obtain the ASCII number corresponding to the character that must be placed. Then, go to the array of data and look at the value corresponding to this character: the value of the variable **h**. Once obtained, the next lines of this loop can be accomplish with no problem. The corrected version of the second loop will be:

```
-----------------------------------------------
     for i = 0 upto (length s) -1:
       sp:= substring(i,i+1) of s;
       if sp <> ‘‘ ’’:
         x:= (long[i] + long[i+1])/2;
         pt:= thelabel(sp, (0,0)) shifted (x,0);
         for j = 16 upto 244:
           if sp = char(j):
             k:= j;
           fi;
         endfor;
```

[4]There are also the files related with the shapes of glyph of little interest for us at this moment.

```
        h:=hig[k];
        draw pt shifted (-xpart pt, h) rotated angle direction
arctime x of pi  of pi shifted point arctime xpart pt of pi  of pi;
      fi;
    endfor;
-----------------------------------------------
```

And the final macro will be the addition of the first loop at the beginning of this second one, plus some inclusions of a general kind.

Compacting it a little bit and writing the complete routine, we have:

```
-----------------------------------------------
def txp(expr s, pat, hig) =
picture pt, pic; string sp, ps; path pi; truecorners:= 1;
pi:= pat shifted(-xpart center pat, -ypart center pat);
for i = length s step -1 until 0:
  ps:= substring(0,i) of s;
  pic:= thelabel(ps, (0,0));
  long[i] = 2*(xpart urcorner pic);
endfor;
for i=0 upto (length s) - 1:
  sp:= substring(i,i+1) of s; if sp <> " ":
  x:= (long[i]+long[i+1])/2;
  pt:= thelabel(sp, (0,0)) shifted (x,0);
  for j = 16 upto 244:
  if sp = char(j): k:=j; fi; endfor;
  draw pt shifted (-xpart pt, alt[k]) rotated angle
direction arctime x of pi  of pi shifted point arctime
xpart pt of pi  of pi;
 fi; endfor;
 enddef;
----------------------------------------------
```

The program —ej.mp— that will call this macro can be, for example:

```
----------------------------------------------
beginfig(10); u=0.25mm; string s; path pat;
input fptmbi8r; defaultfont:="ptmbi8r";
s="This is a first try with the first & simplest form of Txp";
pat:= fullcircle scaled 140 shifted(300u,500u) rotated 180;
txp(s,pat,alt); endfig; end
----------------------------------------------
```

A few things must be said to fully understand the listing:

- The file **fptmbi8r** is the one we have created that contains the font data needed. More about that, later on.
- This file is an array from value 16 to 244, to stablish the necesary coincidence of ASCII codes.
- The elements in the array are called **alt**[ ] and not **hig**.

- The negative sign that precedes —sometimes— the value **xpart** is due to the fact that we are working —sometimes— with the center of the bounding box and we look for the left corner.

So with the sole exception of not knowing exactly what is the file **fptmbi8r** —that is totally independat from the metapost code— we have obtained a first macro to write text along a curve. In this case, a circle. Let's look the result of our small program, using the font Times Italic:

FIGURE 1. The first figure created with TXP

The quality of the image can be analised with some kind of zoom for the screen or in the written paper.

6. THE JOY OF TXP: PARAMETERS

The first version of TXP just obtained although works correctly is clearly limited as it is desirable to have more capabilities, for example: scaling the glyph or the path, being able to begin at any point in the path, placing the glyph over or under the baseline or in the middle, modifying the separation between the characters and so on.

We will show the small modifications needed for obtaining this capabilities and their effect in the final image.

Firstly we will introduce the capability of scaling the glyph. This is the same as scaling the string written. This is just a matter of simply sending the scale wanted from the program to the macro as a new parameter. Once received this value as a new constant that can be called, for example **es**, it will be needed to modify slightly some lines of TXP, as shown:

- Line 1 must take acount of the new parameter.
- Line 6 of the macro must be changed from:

```
6_old     pic:= thelabel(ps, (0,0));           to:
6_new     pic:= thelabel(ps,(0,0)) scaled es;
```

- The same with line 12:

```
12_old    pt:= thelabel(sp, (0,0)) shifted (x,0);
12_new    pt:= thelabel(sp ,(0,0)) scaled es shifted (x,0);
```

- And finally the same with line 15:

```
15_old    draw pt shifted (-xpart pt, alt[k]) rotated angle
direction arctime x of pi  of pi shifted point arctime
xpart pt of pi  of pi;
```

```
15_new    draw pt shifted (-xpart pt,es}* alt[k])
rotated angle direction arctime x of pi  of pi shifted point
arctime xpart pt of pi  of pi;
```

And, also in the program, the call to the macro must include this new parameter. If we make this changes and repeat the same figure as above but increasing the scale to a value close to 2, we will obtain:

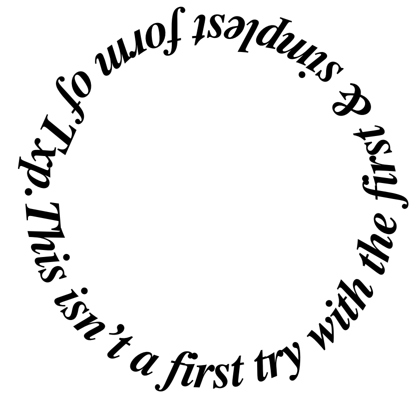

FIGURE 2. Scaling the glyph to the double

For scaling the path, only one *active* line of the macro must be modified, and this is the line 3. Now it must include the efect of this scaling, and if the factor to scale is called **ef**, this line must be written as follows:

```
3_new   pi := pat scaled ef shifted(-xpart center pat*(ef-1),
-ypart center pat*(ef-1));
```

If this line is changed and the corresponding parameters included, maintaining the values given for the last figure, and giving a value to the scale of the path equal to 2, we obtain the figure:

FIGURE 3. Scaling the path to the double

The changes done are clearly visible and the result obtained the expected one.

In this same way and philosophy it is possible to add many new features to the macro, but to maintain this paper in a reasonable size we will not continue in this *step by step* way of increasing the capabilities. At the end we will write a quite complete version of TXP and an example that will show the many capabilities implemented[5]

7. Closing the fonts discussion

At this stage of this paper, only one point has remained in what could be called *the misterious depth*. We are refering to the font data files. A big part of the mistery has been already explained and we have left for the last part the total knowledge because as we said before, really is not a metapost affaire.

What TXP needs is the possibility to acces certain data related with the font geometry. More specifically, the size of the bounding box of every glyph and its position in relation with the baseline. All those data are included in binary form in the files *.tfm as we have said. When we transform these files to the format *.pl, we make readable the content of the .tfm file. So, the file .pl contains all that is needed by TXP. The only thing that remains for giving the data to TXP is just to organize and arrange them. But Metapost is not an ideal language for doing that and a small helping programming language, like AWK is the perfect one. So, we have written a few lines in awk that we wish to show now. Also will be explained the way to uses this very short routine.
The routine gentyp.akw that we are going to comment does a simple but fundamental job. GENTYP looks at all the lines of the .pl file, read some values from the lines that begin with the words: CHARACTER, CHARHT and/or CHARDP, makes a simple calculation and write the result to a helping file called **fontdat**. If we have taken as our .pl file the one corresponding to the font "palatino bold roman", its name would be **pplb8r**. Once obtained the file fontdat we would need to change this name to **fpplb8r**, and this is the file to *serve* to TXP. For any other font, tha way of proceeding would be exactly the same IF the font is of the type "8", so up to 255 possible glyph.
Let's write the listing of this help routine. Here it is gentyp.awk:

```
--------------------------------------------------------------
BEGIN {i=0}
    $1 == "(CHARACTER" { letter[i]=$3; i++}
    $1 == "(CHARHT" { high[i-1]=10*$3;}
    $1 == "(CHARDP" { deph[i-1] = 10*$3 ;}
END{for(j=0;j<229;j++) print "alt["j+15"]:=  \
"(high[j]+deph[j])/2-deph[j] > "fontdat";}
--------------------------------------------------------------
```

[5]We have limited purposely this capabilities to let a reasonable amount of parameters, but it can be increased with no problem. For example, to add the possibility of scaling the text with a different horizontal and vertical scale is trivial, and the same is true for the shearing of the bounding boxes that can furnish an interesting tool for special cases in which it is desired to do something *fantaisiste* as, for example, transforming an italic font in a vertical one or the contrary.

Summarizing, what is needed to create the font data file is just to write:

awk $-$f gentyp.awk pplb8r.pl and then:
mv fontdat fpplb8r

And that's all. Once done this preparatory work for all the fonts usually employed by anyone, he has not need to care anymore.[6]

8. The best at the end

In what follows we will show the final aspect of the txp.mp macro and an example of aplication with different fonts, scales and colors. We will finish by giving a few recomendations of practical aspect.

Final Listing Of Macro txp.mp.-

```
-----------------------------------------------------------
def txp(expr s, pat, es, ef, hi, tr, se, lc, th, hig) =
picture pt ,pic; string sp, ps; color loc; path pi;
truecorners:=1;
pi := pat scaled ef shifted(-xpart center pat*(ef-1), \
-ypart center pat*(ef-1));
long[0]:=tr;
for i = length s step -1 until 1:
  ps:=substring(0,i) of s;
  pic:=thelabel(ps,(0,0)) scaled es;
  long[i]:=((2+se)*(xpart urcorner pic) + tr);
endfor;
for i=0 upto (length s) - 1:
  sp:= substring(i,i+1) of s;
  if sp <> " ":
    x:= (long[i]+long[i+1])/2;
    pt:= thelabel(sp ,(0,0)) scaled es shifted (x,0);
    for j = 16 upto 244:
      if sp = char(j): k:=j; fi; endfor;
     h:=es*(alt[k] + hi);
    draw pt shifted (-xpart pt,h) rotated angle direction \
arctime x of pi  of pi shifted  point arctime xpart pt of pi  \
of pi withcolor lc;
  fi; endfor;
if th <> 0:
  pickup pencircle scaled th;
  draw pi withcolor red;
fi;
enddef;
------------------------------------------------------------
```

With this macro, there are practically no limitations to the capability of writing text strings, as long as wanted —althoug in a single paragraph— using the type

[6]Included with this paper will be the fondat files of three very important families of fonts: Times, Palatino and Helvetica. Also we will include the file corresponding to ZapfChancery.

and size of fonts and the shape of the path wanted. It is possible to mix all those components in a single program as will be shown in the example that follows.

9. An application to finish with

The program that will be shown immediatly gives a quite interesting idea of the possibilities of TXP. Here is the listing:

```
--------------------------------------------------------------
beginfig(1);
input txp; u=0.25mm; color loc; string s; path a;
es:=1; ef:=1; hy:=0; tt:=0; sep:=0; loc:=black; lin=0;

input fptmbi8r;
defaultfont:="ptmbi8r";
s:="WRITING ON THE PATH IS AMUSING AND EASY...WRITING ON THE PATH \
IS AMUSING AND EASY...WRITING ON THE PATH IS AMUSING AND EASY... \
WRITING ON       THE PATH IS AMUSING AND EASY...WRITING ON THE \
PATH IS AMUSING AND EASY...";
a:=(192u,768.0u).. controls (124u,824.0u) and (128u,904.0u).. \
(166u,934.0u).. controls (204u,964.0u) and (268u,944.0u).. \
(320u,886.0u).. controls (372u,828.0u) and (380u,768.0u).. \
(438u,742.0u).. controls (496u,716.0u) and (584u,752.0u).. \
(590u,832.0u).. controls (596u,912.0u) and (548u,952.0u).. \
(504u,974.0u).. controls (460u,996.0u) and (348u,1004.0u).. \
(294u,938.0u).. controls (240u,872.0u) and (260u,776.0u).. \
(330u,732.0u).. controls (400u,688.0u) and (660u,676.0u).. \
(688u,860.0u);
hy:=-3;sep:=0.2;loc:=blue;
txp(s,a,es,ef,hy,tt,sep,loc,lin,alt);

input fpzcmi8r;
defaultfont:="pzcmi8r";
s:="YES !!";
a:=(400u,850u)--(500u,850u);
es:=4;hy:=0.;sep:=0.;loc:=red;
txp(s,a,es,ef,hy,tt,sep,loc,lin,alt);

input fphvb8r;
defaultfont:="phvb8r";
s:="Although sometimes, it can be cumbersome.";
a:=(600u,1000u)..(425u,1100u)..(250u,1000u);
es:=1.3; sep:=.0; loc:=black;
txp(s,a,es,ef,hy,tt,sep,loc,lin,alt);

input fpplbo8r;
defaultfont:="pplbo8r";
s:="But, anyway, I like it !! ...";
a:=(250u,600u)--(600u,600u);
es:=1.8; sep:=0.3;loc:=green;
```

```
txp(s,a,es,ef,hy,tt,sep,loc,lin,alt);

input fphvb8r;
defaultfont:="phvb8r";
s:=":-)";
a:=(400u,500u)--(400u,400u);
es:=4;hy:=0.;sep:=0.;loc:=red;
txp(s,a,es,ef,hy,tt,sep,loc,lin,alt);

input fphvr8r;
defaultfont:="phvr8r";
s:="O";
a:=(355u,500u)--(390u,500u);
es:=11;hy:=-5.4;sep:=0.;loc:=blue;
txp(s,a,es,ef,hy,tt,sep,loc,lin,alt);
endfig; end
---------------------------------------------------------------
```

And here —in the next page— is the result of running it:

We will finish with a mention to the meaning of the parameters used in the general macro, although, as we have already said, this is just a *limited* version.

Parameter and meaning.-

```
s  ---> String to write;
a  ---> Path to write to;
es ---> Scale used for the fonts. (Default: 1; No scaled)
ef ---> Scale used for the path. (Default: 1; No scaled)
hy ---> Vertical placement over the path.
        (Default: 0; The path is the baseline)
tt ---> Distance between the beginning of the written
        string and the beginning of the path. (Default: 0)
sep---> Extra separation between the glyph. (Default: 0)
loc---> Color to use for each string. (Default: black)
lin---> Thickness used to draw the path. (Default: 0: No draw)
```

NOTE: For using Computer Modern fonts of type “7”, some small arrangements must be made. Once understood this paper, it is a very simple matter. If in doubt, visit the Web Page at:

http://w3.mecanica.upm.es/metapost

or contact me directly by email to the address shown.

Departamento de Mecanica, E.T.S. de Ingenieros de Caminos C. y P. (U.P.M.), Ciudad Universitaria. Madrid 28040

E-mail address: smuelas@mecanica.upm.es

URL: http://w3.mecanica.upm.es/~smuelas

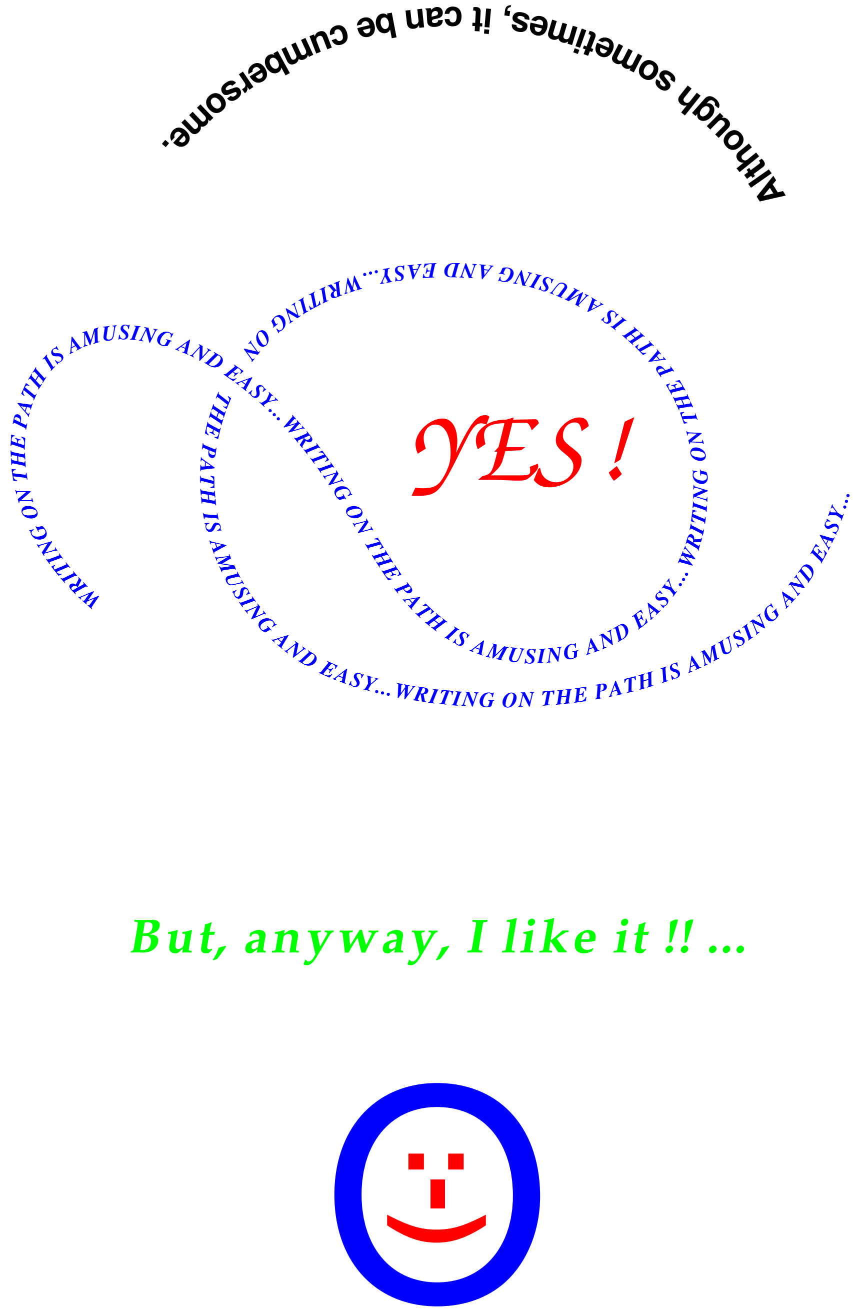

FIGURE 4. The last figure of this paper