Turtle Graphics and TEX — a child can do it

Kees van der Laan
Hunzeweg 57
9893 PB Garnwerd
The Netherlands
Email: `cgl@rc.service.rug.nl`

Abstract

Papert's Turtle Graphics in TEX provide the user with a new method for handling drawings via TEX alone. Being aware of explicit coordinates is replaced by the body language of drawing using the points of a compass. The approach is suited for highly systematic figures such as fractals. Example shapes include spiral, Pythagorean tree, binary tree, and rotated binary tree. The macros are part of BLUe's format system and available from CTAN and the NTG's 4AllTEX CD-ROM.

Introduction

Turtle Graphics has its roots in the pedagogical approach of Piaget. It comes down to learning by metaphors. Computer graphics are demonstrated to children via a turtle[1] moving on the screen. A petal can be drawn by starting at the origin and moving north towards 'up + right', arriving horizontally; then leaving southbound and arriving horizontally at the origin.

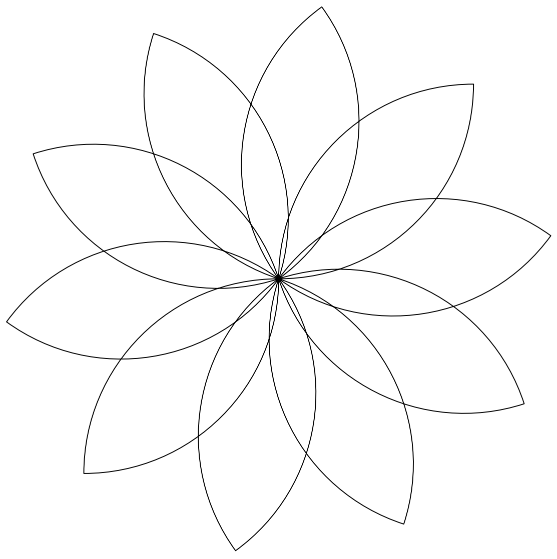
Example *(Flower borrowed from Papert)*

The flower picture is obtained via first creating a basic petal by moving in quarter circles, and then combining several of them. The turtle moves along rotated petals. In METAFONT this is coded essentially as follows:

```
petal=origin{up}..
     {right}(up+right){down}..
     {left}origin;
for k=1 upto 10:
   draw petal rotated36k;
endfor
```

`petal` is a path; the path data structure in METAFONT is powerful.

In this short paper I will discuss what has been used in BLUe's format system[2] as an extension to `manmac` in the turtle graphics macros, especially in the coding of the points of a compass: `\N`, `\E`, `\S`, `\W`, `\NE`, `\SE`, `\SW`, `\NW`, along with `\ESE`, and `\WSW`. I will restrict myself to straight lines in TEX.



Papert's petals

Examples are included which show what can be attained by these basic functionalities. Now and then a METAFONT alternative has been included. In the Appendix some tree diversions have been given.

Why?

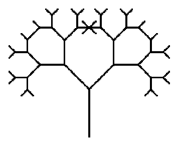
The need for general and flexible line elements arose when I faced the problem of how to draw classical fractals in TEX.[3] The very least is the possibility to draw lines at 45°, the mid-points of a compass.

[1] Knuth already used the turtle idea in his dragon figures (1996, p. 391). For those interested in turtle graphics, consult Papert (1980), for example.

[2] For more information, see my paper, "BLUe's Format — the off-off alternative," elsewhere in these proceedings.

[3] Inspired by Gurari's work on TEX and graphics.

Example *(Pythagorean tree)*



How to do this in TEX? Via LATEX's picture environment? Too clumsy, and cumbersome when changing the order, for example. Via turtle graphics? Definitely. Via PostScript? A possibility.[4] Via METAFONT? Definitely.

However, why not see how far we can get via TEX alone?

And what about the relevancy? I'm very pleased by the spin-off how to typeset binary trees or charts, even rotated, without the use of PostScript. See the Appendix.[5]

What we need is not the Cartesian picture environment approach, but the good old pen-plotter TEXniques, better known in the pedagogical world as Turtle Graphics.

What is the problem?

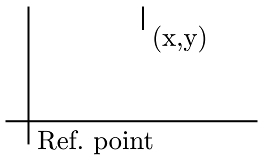
TEX's `\hrule` and `\vrule` primitives are gems and very powerful. It would be nice to have similar primitives for any direction. In the absence of these we can use line pieces provided in fonts. However, the latter suffer from the following drawbacks:

- for a few discrete directions only
- line lengths are discrete too
- line thickness is inflexible

Turtle graphics

The basic idea is that a turtle moves on the screen with the drawing as its trace. How to implement this in TEX?

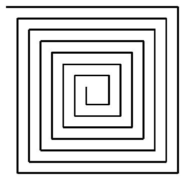
The position of the turtle is maintained in the dimen variables `\x` and `\y`, with TEX's *reference point* left invariant. Moving is parameterized by a direction and by how far to go in that direction. The accompanying figure shows the effect of `\N1`, that is draw the line (`\x`, `\y`) – (`\x`, `\y`+1) — in turtle language the turtle moves up. After completion `\y` has been increased by `\unitlength`.



The movements — our first steps in the turtle graphics world — can be achieved by the following control sequences:

- `\N`, `\E`, `\S`, `\W` mean draw north, east, south and west; similarly, `\NE`, `\SE`, `\SW`, and `\NW`
- `\whiteN`, `\whiteE`, `\whiteS`, `\whiteW` mean *white*-draw north, east, south and west, i.e., the turtle just moves[6]

Example *(Spiral)*
To get the flavor, a classical picture and its coding has been provided, which illustrates that we don't have to worry about coordinates.



The markup reads essentially as follows:

```
\unitlength... \k=1;
\loop\E{\the\k}\advance\k+1
     \S{\the\k}\advance\k+1
     \W{\the\k}\advance\k+1
     \N{\the\k}\advance\k+1
\ifnum\k<29 \repeat
```

More examples have been included in the graphics chapter of the PWT user's guide.

The winds and halfwinds

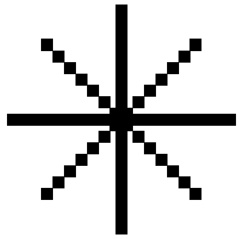
The idea is to compose lines out of elements. I used squares and/or rectangles as elements, and tiled them as follows:[7] For my own reasons, I have also chosen to speak of 'winds' and 'halfwinds' rather than of the points on a compass.

[4] Joseph Romanovsky transcribed my (recursive) code into PostScript.

[5] Or my 'Publishing with TEX' user's guide, PWT for short.

[6] The midpoints can be composed from the four main compass points in this case.

[7] In order to make it visible `\linethickness` has been set to `1ex`. Experiments with bullets and LATEX's line fonts did not yield pleasing results.



The turtle moves `10ex` in each direction, to be tiled by `\hlfwndelm` in the halfwind directions. What is essential is how lines leave a mathematical point. The accompanying model picture has been drawn as follows:

```
\let\0\N \let\1\NE \let\2\E \let\3\SE
\let\4\S \let\5\SW \let\6\W \let\7\NW
\linethickness1ex
\setbox\hlfwndelm=\hbox{\vrule
   width\the\linethickness
   height\the\linethickness depth0pt}
\unitlength10ex
\def\draw{\csname\the\dir\endcsname1}
$$\loop
  \ifnum\dir<8{\draw}\advance\dir1
  \repeat$$
```

Pondering aloud

Can we attain compatibility with TEX's rules primitives? I don't think so, alas.

Thickness. What is meant by thickness if we overlap instead of tile? What is the perceived blackness?

I assumed that tiling with square elements of size `\linethickness`×`\linethickness` — as in the example figure — would yield the same blackness as a rule of thickness `\linethickness`.

Size. Usually the size along the x-axis must be provided. I prefer to have the *real size* specified independently from the orientation of the line. However, the resulting size is not necessarily #1×`\unitlength`.[8] In general the result differs at most by half the atom size because it is composed of a multiple of the basic element. We have to correct by $\sqrt{2}$ to compensate for the direction as we pace along one of the axes. In the example the required length is `10ex`, with as result the tiling of 7 elements of size `1ex`.

For large lines one could think of combining the line elements in LaTeX's `line10` font with the smaller elements. I refrained for two reasons: first, LaTeX's NW line element — `\char'145` — did not fit exactly in the box; and second, because of the inflexibility of the thickness of the font entries.

[8] To put it another way: the required length must be a multiple of the atom size.

Design specs

With the above in mind, I specified the following for the microscopic level — the wind and halfwind commands proper — and for the macroscopic level — the placement within context.

Microscopics. The functionality is to draw a line of the specified length in the direction as implied by the control sequence name. The general specifications read as follows:
- as argument a 'factor' is expected, in order to yield the required length #1×`\unitlength`[9]
- `\linethickness` is a parameter
- after drawing, the position of the turtle is at the end of each line, the reference point has been left invariant, and all the boxes have zero width, height and depth

Extra for the four halfwinds the following:
- `\hlfwndelm` and `\linethickness` are parameters
- draw a line of approximately the specified size
- the atoms are tiled diagonally, at the corners

Macroscopics. Placement within context is the concern of the user. However, because of the zero dimensions of the boxes it is a nuisance to skip or kern when using a picture, in order to create the open space, the niche for the picture. Moreover, when the picture does not take dimensions we are in trouble at page breaks. Therefore assistance is badly needed. The picture environment idea combined with databases comes to the rescue. The *use* of prefab pictures has been simplified in this way, while there is flexibility via `\thispicture` to override the defaults.

Pictures can be stored in BLUe's format `pic.dat` database. Within each database entry the default bounding box and placement within context is provided for. Through the use of `\everypicture` and/or `\thispicture` the defaults can be overridden. This approach complies with the general principles adopted in BLUe's format system.

Coding the winds and halfwinds

It must be emphasized that all boxes have zero dimensions. I also decided to separate getting at

[9] The idea is that not only can integer values be specified but decimal fractions as well.

the (x, y) position from putting whatever there. This is much in the spirit of the second `\point` macro in Knuth (1986, p. 389) and adheres to the *separation of concerns* adage.[10]

In TEX. Familiarity with TEX's *boxes of size zero* is essential: to know the effect of `\kern`-s and `\h/vss`-s inside, and to know the effect of combinations of these boxes.

Kern-s and stretch-or-shrink-s in boxes of size zero.

Example *(Effects of boxes of size zero)*

```
\newdimen\x \x=4ex
\newdimen\y \y=2ex
.\hbox to 0pt{\kern\x a\hss}.
\kern30ex
.\kern\x a.

\noindent and

.\hbox to0pt{\kern\x\vbox to0pt
   {\vss\hbox{a}\kern\y}\hss}.
\kern30ex
.\kern\x\raise\y\hbox{a}.
```

with result

```
     ..   a                    .   a.
and
          a                        a
     ..                        .    .
```

By this mechanism we can move to any point on the page and put there what we wish. Essential is that when a box of zero width is set the *reference point is left invariant* — it is the same before and after.

Putting it together In vertical mode the `\hbox`-es are aligned on the reference point, and when the heights and depths are zero the `\hbox`-es overprint. Moreover, the order of specification is immaterial. In (restricted) horizontal mode `\hbox`-es of width zero overprint and can be given in any order (Knuth 1986, p. 389). In math mode the zero-sized boxes overprint. In display math the invariant reference point is centered horizontally.

Coding After completion the dimension variables `\x` and `\y` have the values of the coordinates of the end of the line. The coding of a few directions has been included to convey the idea.

```
\newbox\hlfwndelm
\newdimen\auxdim %linesize
\newdimen\linethickness
\linethickness1ex
%
\def\xy#1{%Function: place #1 at x, y
   \vbox to0pt{\vss
   \hbox to0pt{\kern\x#1\hss}\kern\y}}
%
\def\xytxt#1{%Function: place text #1
             %          at x, y
   \xy{\vbox to0pt{\vss
   \hbox to0pt{\strut#1\hss
               }\kern0pt}}}
%
\def\N#1{\xy{\kern-.5\linethickness
  \vbox to0pt{\vss
  \hrule height#1\unitlength
   width\linethickness}}%
\advance\y#1\unitlength}
%
\def\S#1{\advance\y-#1\unitlength
         {\N{#1}}}
%
\def\SW#1{\auxdim#1\unitlength
          \correction%sqrt2
\loop\advance\auxdim-\wd\hlfwndelm
\ifdim\auxdim>-.5\wd\hlfwndelm
   \advance\x-\wd\hlfwndelm
   \advance\y-\ht\hlfwndelm
   \xy{\vbox to0pt{\vss
            \copy\hlfwndelm}}%
\repeat}
```

In METAFONT. The coding to go one step north in METAFONT reads as follows:

```
def north=draw z--
    hide(y:=y+size)z enddef;
```

To draw in any direction in METAFONT is implicit, just provide:

```
draw <beginpoint>--<endpoint>
```

There isn't much need to provide turtle graphics macros in METAFONT — it is essentially already there.

Coding the Pythagorean tree

The following illustrates the use of basic turtle movements for this class of problems. Moreover, it shows that coding in TEX is completely different from coding in METAFONT. This goes deeper than a mere difference in notation.

[10] In my view, it makes the code more trustworthy and avoids pitfalls, especially the potential confusion between the kerns needed to get at (x, y) and the kerns to position what has to be put at (x, y).

In TEX. Via the use of the winds and halfwinds the Pythagorean tree code in TEX reads as follows:

```
\def\pythtree{\ifnum\level=1
                    \eerthtyp\fi
    \advance\level-1
    \multiply\kk23\divide\kk32%
    {\leftbranch\draw\pythtree}%
     \rightbranch\draw\pythtree}
\def\eerthtyp#1\pythtree{\fi}
%with auxiliaries
\let\0\N \let\1\NE \let\2\E
\let\3\SE\let\4\S \let\5\SW
\let\6\W \let\7\NW
\def\leftbranch{\advance\dir7
 \ifnum\dir>7 \advance\dir-8 \fi}
\def\rightbranch{\advance\dir1
 \ifnum\dir>7 \advance\dir-8 \fi}
\def\draw{\csname\the\dir\endcsname
         {\the\kk}}
%with use
$$\unitlength0.1pt\kk128 %Size
                   \level5%Order
  \N{\the\kk}             %Trunk
  \pythtree$$
```

Note that there is no build-up of `\fi`-s, no use of either `\expandafter` or `\let` (this last has been used throughout *The TEXbook*).

In METAFONT. The turtle idea has been used in going from node to node in METAFONT as follows:[11]

```
pair node[];
n=15;          %order
l=75;          %size of the trunk
node[0]=origin;%position, and
d= 90;         %orientation trunk
%Create nodes of leftbound branch
for k=1 upto n:
   node[k]=node[k-1]+l*dir d;
   d:=d+45;l:=.7l;
endfor
%Draw the tree
for k=n-1 downto 1:
  draw node[k+1]--node[k];
  addto currentpicture also
  currentpicture rotatedaround
               (node[k],-90);
endfor
draw node1--node0;
drawdot origin; showit
end
```

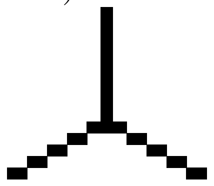
[11] But ...only for the left branch; to draw the other branches the symmetry operation — `rotatedaround` — has been used.

Note that there is no recursion. The symmetry operations of METAFONT allow a concise implementation, with much faster performance than when all the leaves would have been walked through one after another.[12]

Trinaries

For 45° lines I used square elements. Why not use rectangular elements for 30° lines in conformance with the direction?

Example *(Lines at 30°)*



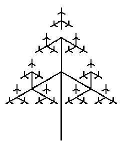
This model has been obtained as follows:

```
$$\x0pt\y0pt
  {\N{10}}{\ESE{10}}{\WSW{10}}$$
%with initializations
\linethickness1ex
\setbox\trielm=\hbox{\vrule
   width1.74\linethickness
   height\linethickness\relax}
%To account for element in 30 degrees
%direction
\unitlengthy\ht\trielm %default.2pt
\unitlengthx\wd\trielm %default.3482pt
\unitlength\unitlengthy%default.2pt
%and the macros
\def\WSW#1{\auxdim#1\unitlength
           \divide\auxdim2
   \loop\advance\auxdim-\unitlengthy
   \ifdim\auxdim>-.5\unitlengthy
      \advance\x-\unitlengthx
      \advance\y-\unitlengthy
      \xy{\vbox to0pt{\vss
                     \copy\trielm}}%
   \repeat}
%
\def\ESE#1{\auxdim#1\unitlength
           \divide\auxdim2
   \loop\advance\auxdim-\unitlengthy
   \ifdim\auxdim>-.5\unitlengthy
      \advance\y-\unitlengthy
      \xy{\vbox to0pt{\vss
                     \copy\trielm}}%
```

[12] In the seventies these kinds of problems had a reputation of keeping pen-plotters busy. Because of raster devices we can now do much better, and METAFONT allows us to prescribe this.

```
        \advance\x\unitlengthx
    \repeat}
```

Example *(Trinary tree)*



```
$$\x0pt\y0pt\level6 \kk128
  \N{128}\tritree$$
%with trinary tree macro
\def\tritree{\ifnum1=\level
                     \eertirt\fi
    \advance\level-1 \divide\kk2
    {\N{\the\kk}\tritree}%
    {\ESE{\the\kk}\tritree}%
     \WSW{\the\kk}\tritree}
\def\eertirt#1\tritree{\fi}
```

Remark: The `\unitlength`-s are, by default, equal to the sides of the elementary rectangular block. The size of the tree can be controlled by `\kk`.

Coding a database element

When inserting a picture in BLUe's format picture database, extra layers are added to the picture code to allow for reuse and to parameterize scalability, positioning, visibility, with defaults provided, and to set a picture within a box of the right size, the bounding box.

How to create a database element has been treated elsewhere and is not repeated here. However, I have included an example to convey the idea.

Example *(The database element* `bintreepic`*)*
The `\bintreepic` element of the `pic.dat` database reads as follows:

```
\lst\bintreepic{\bgroup
   \unitlength.5ex\kk32
   \xoffset{-32} \yoffset{-2}%
   \xdim{66}\ydim{5}%
   \def\eertnib##1\bintree{\fi}
   \beginpicture\bintree\endpicture
   \egroup\thispicture{}}
%with in the kernel blue.tex
\def\bintree{\S1\ifnum\kk=2
                \eertnib\fi
   \divide\kk2
   {\W{\the\kk}\bintree}%
    \E{\the\kk}\bintree}
%and accounting for the leaves
\def\eertnib#1\bintree{\fi
   \global\advance\k1
   \whiteS1\xytxt{
   \csname\the\k\endcsname}}
```

Explanation: `\bintreepic` comes down to an invocation of `\bintree` with scaling and positioning parameters added, assigned with default values. The defaults can be overridden via the use of `\thispicture{...}`. The tokens provided by the latter are inserted by `\beginpicture`.

Epilogue

The lines at 45° have little compatibility with TEX's rules, alas, especially with non-neglible thickness. I was surprised to realize that TEX's defaults for rules are not symmetric around their axes in relation to the reference point.

Have fun, and all the best.

References

Gurari, Eitan M. *TEX and LATEX: Drawing and Literate Programming*. New York: McGraw-Hill, 1994.

Knuth, D.E. *The TEXbook*. Reading, MA: Addison-Wesley, 1986.

Papert, S. *Mindstorms; Children, Computers, and Powerful Ideas*. New York: Basic Books, 1980.

van der Laan, K. Publishing with TEX: BLUe's Selection. Garnwerd, Holland, 1995. Available via CTAN (`info/pwt`).

van der Laan, K. "BLUe's Format — the off-off alternative." Elsewhere in these *Proceedings*.

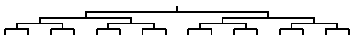
Appendix: Binary tree and chart

Example *(Binary tree)*

```
\pictures\bintreepic

$$\bintreepic$$
```

with result



Rotated tree. Once we understand turtle graphics, rotating a tree can be done easily by shifting the meaning of the directions, and adjusting the positioning of the leaves.[13] In the code below, `\bintree` and `\eertnib` come with `blue.tex`, and `\rotatedbintreepic` is included in `pic.dat`. The `\rotatedbintreepic` entry reads as follows:

```
\lst\rotatedbintreepic{%
\bgroup\unitlength1ex%
  \let\W\N \let\exchange\E
  \let\E\S \let\S\exchange
  \def\1{x}\def\2{y}\def\3{a}
  \def\4{b}\def\5{piet}%
  \def\6{hans}\def\7{etc.}%
  \k0\kk16\xdim{10}\ydim{30}%
\beginpicture\bintree\endpicture
\egroup\thispicture{}}
```

Example *(Rotated tree)*

```
\thispicture{\def\1{cgl}
  \def\2{PWT}\def\3{July}
  \def\4{1995}\def\5{\dots}
  \def\6{}\def\7{}
  \yoffset{-16}\ydim{28}}
$$\rotatedbintreepic$$
```

yields

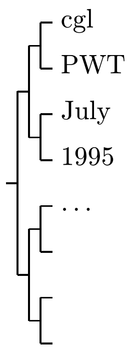
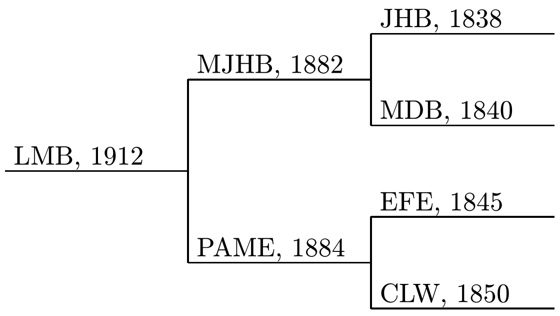


Chart. Through the `\bintree` macro we can also obtain charts elegantly.

[13] A white lie. The tree is actually mirrored because I like the leaves to be numbered from the top. In general we can rotate via PostScript.

Example *(Chart – The TEXbook, p. 248, ex. 22.14)*



obtained via

```
%labels in preorder
%(default in \chartpic)
\def\1{LMB, 1912}
\def\2{MJHB, 1882}\def\5{PAME, 1884}
\def\3{JHB, 1838} \def\4{MDB, 1840}
\def\6{EFE, 1845} \def\7{CLW, 1850}
\ekk8
\k0\unitlength2ex\x0pt\y0pt\kk8
\hbox{\modbintree}
%with auxiliaries
\let\Eold\E
\def\E{\global\advance\k1
  \xytxt{
  \csname\the\k\endcsname}\Eold}
```

Remarks: An aid in finding the numbers of the branches is to delete `\csname` and `\endcsname` in `\E`. The way of traversal at hand is called preorder.

When using `\chartpic` from `pic.dat` the texts along the branches — `\def\1{...} etc.` — have to be supplied as tokens within a `\thispicture`. And one final note: `\modbintree` is the adjusted `\bintree` macro for this case.