

Slanted lines with controlled thickness

David Salomon and Matthew N. Hendryx

So far as the authors know, the original idea for a slanted line is due to A. Hendrickson (Ref. 1). The idea was to typeset a period, to move a small step in the desired direction, and to repeat the process. Some improvements to the basic idea are described in Ref. 2, together with the observation that better results can be obtained by typesetting a small rule, since the size of a rule can be precisely controlled, and can be adapted to the specific printer used.

`PfCTeX`, by M. Wichura (Ref. 3), is an excellent macro package for drawing diagrams in `plain TeX`. It uses the basic idea of typesetting a period and moving it. The article by Wichura does not explain the plotting algorithms used by `PfCTeX`, except to say that linear and quadratic interpolation are used.

One problem with the traditional method is the lack of control over the thickness of the resulting line. In a high-quality diagram containing horizontal, vertical and slanted lines, the thickness of all lines should be the same.

The principle

The method described here makes it possible to typeset slanted lines of any thickness by typesetting a rule, shifting it in the desired direction, and repeating the process a number of times (Fig. 1).

The user specifies the following four quantities:

- 1-2. X and Y , the horizontal and vertical displacements of the line, respectively (Fig. 1).
3. The thickness, t , of the line (Fig. 2).
4. The height, h , of a rule. This is determined by the printer used. For a given printer, the same value of h is used for all slanted lines.

The program has to calculate three values:

1. The width, b , of a single rule (Fig. 1).
2. The amount i by which each rule is shifted relative to its predecessor (Fig. 1).
3. The number r of rules necessary to get a complete slanted line (register `\rep` in the macros below).

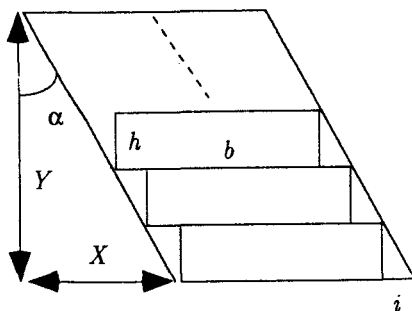


Fig. 1

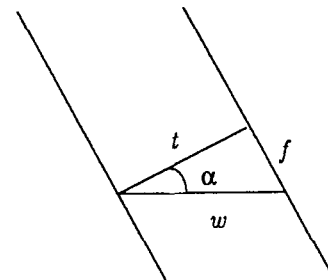


Fig. 2

The derivation

The three quantities are derived from elementary trigonometry. From Fig. 1 we get $\tan \alpha = X/Y$ and also $\tan \alpha = i/h$ or $i = h \tan \alpha$. From Fig. 2, $f^2 + t^2 = w^2$ and also $\sin \alpha = f/w$ or $f = w \sin \alpha$ which, in turn, implies $w^2 \sin^2 \alpha + t^2 = w^2$ or $t^2 = w^2(1 - \sin^2 \alpha) = w^2 \cos^2 \alpha = w^2/(1 + \tan^2 \alpha)$. (The last step uses the identity $\sin^2 \alpha + \cos^2 \alpha = 1$.) Since w is defined as $b + i$, we get $b = t\sqrt{1 + \tan^2 \alpha} - i$. The number r of necessary rules is easily seen to be $(Y/h) - 1$.

To summarize, the quantities X , Y , t , and h are given. From them, the three quantities i , b and r should be calculated by: $i = h \tan \alpha = hX/Y$, $b = t\sqrt{1 + \tan^2 \alpha} - i = t\sqrt{1 + (X/Y)^2} - i$ and $r = (Y/h) - 1$.

The only problem is the square root calculation. This calculation is easy to perform using Newton's method but, because of the limited precision of `TeX`, the results are often imprecise (which does not seem to affect the quality of the final lines by much). Here are the details of Newton's method.

Square root calculation

Newton's method for finding a root of a given function $f(x)$, is iterative. One starts with a first approximation x_0 (usually a guess), and performs the iteration $x_{i+1} \leftarrow x_i - f(x_i)/f'(x_i)$, $i = 0, 1, \dots$

To adapt the method for square root calculation, we select the function $f(x) = x^2 - n$. Clearly, any root of this function equals \sqrt{n} . Since $f'(x) = 2x$, the iterations above become

$$x_{i+1} \leftarrow x_i - \frac{x_i^2 - n}{2x_i} = \frac{1}{2} \left(x_i + \frac{n}{x_i} \right), \quad i = 0, 1, \dots$$

A good guess for x_0 is $n/2$, and 3 or 4 iterations are usually sufficient to get within 1% of the right value. We use eight iterations, to get better precision for small values of n .

Since the calculations involve non-integers, a `TeX` implementation should use `\dimen` registers. Since the calculations involve division, `\count` registers are also necessary. In the macros below, the `\dimen` register `\nn` stands for n , `\xx` stands for x_i

and $\backslash yy$, for x_{i+1} . The final result is returned in $\backslash yy$. The calculation is straightforward except for two points:

1. The ‘ $\backslash multiply\backslash yy$ by 100’ is necessary since otherwise the division that follows ($\backslash divide\backslash yy$ by $\backslash xx$) would result in a truncated quotient.

2. The ‘ $\backslash multiply\backslash yy$ by 655’ is necessary to scale the value of $\backslash yy$ from scaled points to points. The full factor is 65536 but we use 655 because of the previous multiplication by 100. The macros are:

```
\newdimen\mn \newcount\xx \newdimen\yy
\def\sqrt#1{\nn=#1 \xx=\nn \divide\xx by2
\iter\iter\iter\iter\iter\iter\iter\iter}
\def\iter{\yy=\nn \multiply\yy 100
\divide\yy by\xx \multiply\yy by 655
\advance\yy by\xx sp \divide\yy by2
\xx=\yy}
```

A typical expansion is $\backslash sqrt\{.8in\}$. Following which, the command ‘ $\backslash the\backslash yy$ ’ produces ‘7.59865pt’, less than 1% away from the true value. Note that, because of the limited arithmetic capabilities of T_EX, values over 163pt cause an arithmetic overflow. For small values, more iterations may be necessary. For example $\backslash sqrt\{50sp\}$ produces 0.125pt after 4 iterations, 0.036pt after 6 iterations, and 0.024pt after 8 iterations. The correct value is close to 0.0276pt. A more robust square-root macro is presented in a later section.

The result

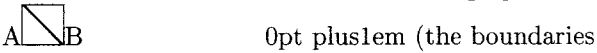
The final macro, $\backslash slant$, takes three parameters, the quantities X , Y and t above. It creates the rules in $\backslash box\backslash slnt$ whose width is (Fig. 1) $X + b + i$ and whose height is Y . The user can then typeset the box in any desired way.

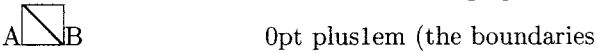
Note that the height of an individual rule is not a parameter of $\backslash slant$ but must be assigned explicitly to the $\backslash dimen$ register $\backslash hh$ before $\backslash slant$ is expanded. This is because our experience shows that, in practice, the height of the rules that make up the slanted lines depends on the printer used, and is thus the same for all slanted lines in the document. It is easy, of course, to specify the height as a parameter (#4) of $\backslash slant$, if desired. The macro should simply say ‘ $\backslash ttt=#3 \backslash ii=#2 \backslash auxi=#4$ ’.

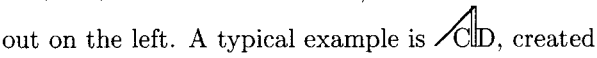
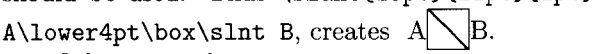
The macro has four parts. The first three calculate r (in register $\backslash rep$), i and b . Part four creates the rules.

```
\newdimen\ttt \newdimen\hh \newdimen\ii
\newdimen\bb \newdimen\tga
\newcount\auxi \newcount\rep \newbox\slnt

% Part 1. Number of repetitions
\def\slant#1#2#3{\ttt=#3 \ii=#2 \auxi=\hh
\multiply\ii by655 \divide\ii by\auxi
\multiply\ii by100
\rep=\ii \divide\rep by65536 % \rep:=Y/h-1
% Part 2. i=h tan a
\tga=#1 \ii=#2 \auxi=\ii
\multiply\tga by655 \divide\tga by\auxi
\multiply\tga by100 \auxi=\tga \ii=\hh
\divide\ii by655 \multiply\ii by\auxi
\divide\ii by100 % i=h tan a
% Part 3. b:=t\sqrt{tan^2a +1}-i
\divide\auxi by10 \multiply\auxi by\auxi
\divide\auxi by655 % tan^2a
\tga=\auxi sp \advance\tga by1pt
\sqrt{\the\tga}% \sqrt{tan^2a +1}
\auxi=\yy \divide\auxi by655 \bb=\ttt
\multiply\bb by\auxi \divide\bb by100
\ifdim\ii>0pt\advance\bb by-\ii
\else\advance\bb by\ii\fi % b:=t\sqrt{..}-i
\tga=0pt % Part 4. Build the rules
\setbox\slnt=\vbox{\offinterlineskip
\loop \ifnum\rep>0\advance\rep-1
\hbox{\kern\tga\vrule width\bb height\hh}%
\advance\tga by\ii
\repeat}}
```

A typical expansion is: $\backslash hh=.3pt$
 $\backslash slant\{15pt\}\{15pt\}\{1pt\}A\backslash box\backslash slnt B$,
 which results in  Opt plus 1em

 Opt plus 1em (the boundaries of $\backslash box\backslash slnt$ are shown for illustration purposes). The macro works for a negative X , but the width of $\backslash box\backslash slnt$ in this case is b , and the line sticks

out on the left. A typical example is  CD , created by $\backslash slant\{-15pt\}\{15pt\}\{1pt\}C\backslash box\backslash slnt D$. The macro does not work for negative values of Y . To create slanted lines that go below the text, a $\backslash lower$ should be used. Thus $\backslash slant\{15pt\}\{15pt\}\{1pt\}A\backslash lower4pt\backslash box\backslash slnt B$, creates  B .

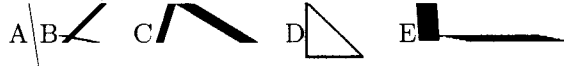
Other examples are:

```
\hh=.3pt
A\slant{4pt}\{25pt\}\{.4pt\}\lower10pt\backslash box\backslash slnt
B\slant{15pt}\{3pt\}\{1pt\}\backslash box\backslash slnt
\slant{-15pt}\{15pt\}\{2pt\}\backslash box\backslash slnt
\quad
C\kern6pt\slant{-5pt}\{15pt\}\{3pt\}\backslash box\backslash slnt
\slant{25pt}\{15pt\}\{4pt\}\backslash box\backslash slnt
\quad
```

```
\hh=.5pt
D\lower6pt\hbox{\vrule height21pt width1pt
 \slant{21pt}{21pt}{1pt}\wd\slnt=0pt\box\slnt
 \vrule width21pt height1pt depth0pt}
\quad
```

```
E\slant{1pt}{15pt}{8pt}\box\slnt
 \slant{15pt}{3pt}{8pt}\box\slnt
```

resulting in



Note that the last example is wrong. We ask for a slanted line with $Y = 3\text{pt}$ and a thickness of 8pt , which is impossible. The line came out with a thickness of 3pt . The reader should compare this line with, e.g., `\slant{15pt}{13pt}{8pt}\box\slnt`, which has the right thickness of 8pt .

Large values of `\hh` can create nice patterns (try `\hh=4pt\slant{18pt}{18pt}{6pt}\box\slnt`), but experience shows that values around 0.3pt – 0.5pt are best for a typical 300dpi laser printer.

Possible improvements

1. The square root algorithm used here is iterative. It turns out that, for lines with slants close to 45° , 3 or 4 iterations are enough to get to within 1% of the correct square root. For slants closer to 0° or to 90° , more iterations are necessary. In a document with many slanted lines, it is possible to speed up the macros by changing the number of iterations depending on the slant, because both $\tan \alpha$ and $\cot \alpha$ are available.

2. The square-root macro presented earlier is simple and fast, but is not robust. It causes an arithmetic overflow for values over 163pt . The macro shown below is slower and more complex, but produces results accurate to 28 significant bits. This macro is the one used by METAFONT to calculate square-roots (Ref. 4, sections 121–123). It has been translated from WEB to T_EX, and has been provided to us by the referee.

```
\def\incr#1{\advance#1 by 1 }
\def\decr#1{\advance#1 by -1 }
\def\half#1{\divide#1 by 2 }
\def\double#1{\multiply#1 by 2 }
\def\fractiontwo{536870912 } % 2^29,
% represents 2.000000000
\def\fractionfour{1073741824 } % 2^30,
% represents 4.000000000

% Find the square root of #1 placing the
% results in \yy.
% The square root is in scaled numbers
% which are integer representations
```

```
% of numbers multiplied by 2^{-16}.
% That is, 1 = 65536, 2=131072,
% and so on. This uses Newton
% approximation, with shifts to
% preserve accuracy.
```

```
\newcount\k \newcount\y
\newcount\q \newcount\x
\newdimen\yy
```

```
\def\sqrt#1{\yy=#1 \x=\yy
 \ifnum\x>0
 \k=23 \q=2
 \loop\relax
 \ifnum\x<\fractiontwo
 \decr\k \multiply\x by 4
 \repeat
 %
 \ifnum\x<\fractionfour \y=0
 \else
 \advance\x by -\fractionfour \y=1
 \fi
 %
 % Decrease k by 1, maintaining the
 % invariant relations between x, y & q
 \loop
 \double\x \double\y
 \ifnum\x<\fractionfour \else
 \advance\x by -\fractionfour
 \incr\y
 \fi
 \double\x
 \advance\y by \y \advance\y by -\q
 \double\q
 \ifnum \x<\fractionfour \else
 \advance\x by -\fractionfour
 \incr\y
 \fi
 \ifnum\y>\q
 \advance\y by -\q \advance\q by 2
 \else\ifnum \y > 0 \else
 \advance\q by -2 \advance\y by \q
 \fi\fi
 \decr\k
 \ifnum\k>0\relax
 \repeat
 %
 \half\q \global\yy=\q sp %final result
 %
 \else
 % case of non-positive argument
 \ifnum\x<0
 \message
 {sqrt of #1 has been replaced by 0}
```

```
\fi
\global\yy=0 sp
\fi}}
```

References

1. Hendrickson, A., *Some Diagonal Line Hacks*, TUGboat **6**(2), 83–86 (July 1985).
2. Salomon, D., *DDA Methods in T_EX*, TUGboat **10**(2), 207–217 (July 1989).
3. Wichura, M., *PiCT_EX: Macros for Drawing Pictures*, TUGboat **9**(2), 193–197 (July 1988). (The actual macros are available from the various T_EX archives.)
4. Knuth, D., *The METAFONTbook*, Reading, MA, Addison-Wesley, 1986.

- ◊ David Salomon
California State University,
Northridge
Computer Science Department
Northridge, CA 91330-8281
dxs@secs.csun.edu
- ◊ Matthew N. Hendryx
P. O. Box 218
North Manchester, IN 46962

Letters

On indexing and errors

In his letter (*TUGboat* 14, no. 2, page 141) Lincoln Durst finds fault with my article (*TUGboat* 13, no. 4, page 495) which in turn found fault in his earlier article (*TUGboat* 12, no. 2, pages 248–52).

However, I still hold to my original criticism, which was that the code of Durst would on specified occasions produce 0010 in an index file when in fact 010 is required. No criticism of Knuth's code was intended, nor I believe made, in my article.

It is easy to make an error. Harder is to find error in the work of another, and harder yet in one's own work. But most difficult, I have found, is to express and accept criticism in a friendly and respectful manner.

Yours sincerely,
Jonathan Fine
203 Coldhams Lane
Cambridge CB1 3HY, England
J.Fine@pmms.cam.ac.uk