Visual Debugging in \TeX

how things are done

Hans Hagen

September 25 1996

Although an integral part of CONTEXT, this module is one of the support modules. Its stand alone character permits use in PLAIN \TeX or \TeX based macropackages. If in some examples the verbatim listings don’t show up nice, this is due to processing by a system that does not support buffering. In CONTEXT we show the commands in the margin, use bit more advanced way of numbering, and typeset the source in \TeXnicolored verbatim. Sorry for this inconvenience.

This module is still in development. Depending on my personal need and those of whoever uses it, the macros will be improved in terms of visualization, efficiency and compatibility.

One of the strong points of \TeX is abstraction of textual input. When macros are defined well and do what we want them to do, we will seldom need the tools present in What You See Is What You Get systems. For instance, when entering text we don’t need rulers, because no manual shifting and/or alignment of text is needed. On the other hand, when we are designing macros or specifying layout elements, some insight in \TeX’s advanced spacing, kerning, filling, boxing and punishment abilities will be handy. That’s why we’ve implemented a mechanism that shows some of the inner secrets of \TeX.

In this module we are going to redefine some \TeX primitives and PLAIN macro’s. Their original meaning is saved in macros with corresponding names, preceded by normal. These original macros are (1) used to temporary restore the old values when needed and (2) used to prevent recursive calls in the macros that replace them.

There are three types of boxes, one horizontal and two vertical in nature. As we will see later on, all three types are to be handled according to their orientation and baseline behavior. Especially \vtop’s need our special attention.

Next come the flexible skips, which come in two flavors too. Like boxes these are handled with \TeX primitives.

Both penalties and kerns are taken care of by mode sensitive primitives. This means that when making them visible, we have to take the current mode into account.

Glues on the other hand are macro’s defined in PLAIN \TeX. As we will see, their definitions make the implementation of their visible counterparts a bit more \TeXnical.

Math mode has its own spacing primitives, preceded by \math. Due to the relation with the current font and the way math is typeset, their unit \mu is not compatible with other dimensions. As a result, the visual appearance of these primitives is kept primitive too.

Fills can be made visible quite easy. We only need some additional negation macros. Because PLAIN \TeX only offers \hfilneg and \vfilneg, we define our own alternative double l’ed ones.
The positive stretch primitives are used independent and in combination with \leaders.

Visualization is not always wanted. Instead of turning this option off in those (unpredictable) situations, we just redefine a few PLAIN macros.

\def\rlap#1{\normalhbox to \!!zeropoint{#1\normalhss}}
\def\llap#1{\normalhbox to \!!zeropoint{\normalhss#1}}
\def\~{\normalpenalty\!!tenthousand\ }

Ruled boxes can be typeset in many ways. Here we present just one alternative. This implementation may be a little complicated, but it supports all three kinds of boxes. The next command expects a ⟨box⟩ specification, like:

\makeruledbox0

We can make the baseline of a box visible, both dashed and as a rule. Normally the line is drawn on top of the baseline, but a smashed alternative is offered too. If we want them all, we just say:

\baselineruletrue
\baselinefilltrue
\baselinesmashtrue

At the cost of some overhead these alternatives are implemented using \if’s:

\newif\ifbaselinerule \baselineruletrue
\newif\ifbaselinefill \baselinefillfalse
\newif\ifbaselinesmash \baselinesmashfalse

Rules can be turned on and off, but by default we have:

\topruletrue
\bottomruletrue
\leftruletrue
\rightruletrue

As we see below:

\newif\iftoprule \topruletrue
\newif\ifbottomrule \bottomruletrue
\newif\ifleftrule \leftruletrue
\newif\ifrightrule \rightruletrue

The width in the surrounding rules can be specified by assigning an appropriate value to the dimension used. This module defaults the width to:

\boxrulewidth=.2pt

Although we are already low on ⟨dimensions⟩ it’s best to spend one here, mainly because it enables easy manipulation, like multiplication by a given factor.
The \cleaders part of the macro is responsible for the visual baseline. The \normalhfill belongs to this primitive too. By storing and restoring the height and depth of box \#1, we preserve the mode.

```
18 40 \def\makeruledbox#1{
19 \edef\ruledheight \the\ht#1
20 \edef\ruleddepth \the\dp#1
21 \edef\ruledwidth \the\wd#1
22 \setbox\scratchbox=\normalvbox
23 {%dontcomplain
24 \offinterlineskip
25 \hrule
26 \!!height\boxrulewidth
27 \iftoprule\else\!!width\!!zeropoint\fi
28 \normalvskip\boxrulewidth
29 \normalhbox to \ruledwidth
30 {\vrule
31 \!!height\ruledheight
32 \!!depth\ruleddepth
33 \iftoprule\else\!!zeropoint\fi
34 \normalvskip\ruleddepth
35 \normalhbox
36 {\vrule
37 \!!width\ruledwidth
38 \iftoprule\else\!!zeropoint\fi
39 \normalvskip\ruledwidth
40 \hrule
41 \!!height\boxrulewidth
42 \iftoprule\else\!!width\!!zeropoint\fi
43 \normalvskip\boxrulewidth
44 \normalhbox to \ruledwidth
45 {\vrule
46 \!!height\ruledheight
47 \!!depth\ruleddepth
48 \iftoprule\else\!!zeropoint\fi
49 \normalvskip\boxrulewidth
50 \normalhbox
51 \cleaders
52 \cleaders
53 \cleaders
54 \cleaders
55 \cleaders
56 \cleaders
57 \cleaders
58 \cleaders
59 \cleaders
60 \cleaders
61 \fi
62 \fi
63 \fi
64 \fi
65 \fi
66 \fi
67 \fi
68 \fi
69 \fi
70 \fi
71 \fi
72 \fi
73 \fi
74 \fi
75 \fi
76 \fi
77 \fi
78 \fi
79 \fi
80 \fi
81 \fi
82 \fi
83 \fi
84 \fi
85 \fi
86 \fi
87 \fi
88 \fi
89 \fi
90 \fi
91 \fi
92 \fi
93 \fi
94 \fi
95 \fi
96 \fi
97 \fi
```

Just in case one didn’t notice: the rules are in fact layed over the box. This way the contents of a box cannot visually interfere with the rules around (upon) it. A more advanced version of ruled boxes can be found in one of the core modules of CONTeXT. There we take offsets, color, rounded corners, backgrounds and alignment into account too.
These macro’s can be used instead of \hbox, \vbox and \vtop. They just do what their names state. Using an auxiliary macro would save us a few words of memory, but it would make their appearance even more obscure.

\begin{verbatim}
\def\ruledhbox{\normalhbox\bgroup\dowithnextbox{\makeruledbox\nextbox\box\nextbox\egroup}\normalhbox}
\def\ruledvbox{\normalvbox\bgroup\dowithnextbox{\makeruledbox\nextbox\box\nextbox\egroup}\normalvbox}
\def\ruledvtop{\normalvtop\bgroup\dowithnextbox{\makeruledbox\nextbox\box\nextbox\egroup}\normalvtop}
\end{verbatim}

Of the next two macros the first can be used to precede a box of ones own choice. One can for instance prefix boxes with \ruledbox and afterwards — when the macro satisfy the needs — let it to \relax.

\ruledbox\hbox{What rules do you mean?}

The macro \setruledbox can be used to directly rule a box.

\setruledbox12=\hbox{Who’s talking about rules here?}

At the cost of some extra macros we can implement a variant that does not need the =, but we stick to:

\begin{verbatim}
\def\ruledbox{\dowithnextbox{\makeruledbox\nextbox\box\nextbox}}
\def\setruledbox#1{\dowithnextbox{\makeruledbox\nextbox\setbox#1=\nextbox}}
\end{verbatim}

Before we meet the visualizing macro’s, we first implement ourselves some handy utility ones. Just for the sake of efficiency and readability, we introduce some status variables, that tell us a bit more about the registers we use:

\begin{verbatim}
\newif\ifflexible
\newif\ifzero
\newif\ifnegative
\newif\ifpositive
\end{verbatim}

These status variables are set when we call for one of the investigation macros, e.g.

\investigateskip|\scratchskip

We use some dirty trick to check stretchability of \textit{skips}. Users of these macros are invited to study their exact behavior first. The positive and negative states both include zero and are in fact non-negative (\(\geq 0\)) and non-positive (\(\leq 0\)).

\begin{verbatim}
\newif\ifflexible
\newif\ifzero
\newif\ifnegative
\newif\ifpositive
\newif\ifflexible
\newif\ifzero
\newif\ifnegative
\newif\ifpositive
\end{verbatim}
123 \ifx\!!stringa\!!stringb \flexiblefalse \else \flexibletrue \fi
124 \ifdim#1=\!!zeropoint\relax
125 \zerotrue \else
126 \zerofalse \fi
127 \ifdim#1<\!!zeropoint\relax
128 \positivefalse \else
129 \positivetrue \fi
130 \ifdim#1>\!!zeropoint\relax
131 \negativefalse \else
132 \negativetrue \fi\)

26133 \def\investigatecount#1%
26134 {\relax
26135 \flexiblefalse
26136 \ifnum#1=0
26137 \zerotrue \else
26138 \zerofalse \fi
26139 \ifnum#1<0
26140 \positivefalse \else
26141 \positivetrue \fi
26142 \ifnum#1>0
26143 \negativefalse \else
26144 \negativetrue \fi}

27145 \def\investigatemuskip#1%
27146 {\relax
27147 \edef\!!stringa{\the\scratchmuskip}\%
27148 \edef\!!stringb{0mu}\%
27149 \edef\!!stringc##1##2\{##1\%
27150 \expandafter\edef\expandafter\!!stringc
27151 \expandafter\!!stringc\!!stringa\%
27152 \edef\!!stringd{-}\%
27153 \flexiblefalse
27154 \ifx\!!stringa\!!stringb
27155 \zerotrue
27156 \negativefalse
27157 \positivefalse
27158 \else
27159 \zerofalse
27160 \ifx\!!stringc\!!stringd
27161 \positivefalse
27162 \negativetrue
27163 \else
27164 \positivetrue
27165 \negativefalse
27166 \fi
27167 \fi}

Indentation, left and/or right skips, redefinition of \par and assignments to \everypar can lead to unwanted results.
We can therefore turn all those things off with \dontinterfere.

28168 \def\dontinterfere\%
28169 {\everypar = {}\%
28170 \let\par = \endgraf
28171 \parindent = \!!zeropoint
28172 \parskip = \!!zeropoint
28173 \leftskip = \!!zeropoint
28174 \rightskip = \!!zeropoint
28175 \relax)}

In this module we do a lot of box manipulations. Because we don’t want to be confronted with too many over- and underfull messages we introduce \dontcomplain.

29176 \def\dontcomplain\%
29177 {\hbadness = \!!tenthousand
29178 \hfuzz = \maxdimen
29179 \vbadness = \!!tenthousand
29180 \vfuzz = \maxdimen}

Now the necessary utility macros are defined, we can make a start with the visualizing ones. The implementation of these macros is a compromise between readability, efficiency of coding and processing speed. Sometimes we do in steps what could have been done in combination, sometimes we use a few boxes more or less then actually needed, and more than once one can find the same piece of rule drawing code twice.
Depending on the context, one can force visual vertical cues being centered along \hsize or being put at the current position. Although centering often looks better, we’ve chosen the second alternative as default. The main reason for doing so is that often when we don’t set the \hsize ourselves, \TeX{} takes the value of the surrounding box. As a result the visual cues can migrate outside the current context.

This behavior is accomplished by a small but effective auxiliary macro, which behavior can be influenced by the boolean \centeredvcue. By saying
\[\centeredvcue{}\]

one turns centering on. As said, we turn it off.

\[\newif\ifcenteredvcue \centeredvcuefalse\]
\[\def\normalvcue#1\fi{\normalhbox \ifcenteredvcue to \hsize \fi \normalhss#1\normalhss}\]

We could have used the more robust version
\[\def\normalvcue\fi{\normalhbox \ifcenteredvcue to \hsize \fi \bgroup\bgroup\normalhss \aftergroup\normalhss\aftergroup\egroup \let\next=}\]

or the probably best one:
\[\def\normalvcue\fi{\hbox \ifcenteredvcue to \hsize \bgroup\bgroup\normalhss \aftergroup\normalhss\aftergroup\egroup \else \bgroup \fi \let\next=}\]

Because we don’t have to preserve (catcodes) and only use small arguments, we stick to the first alternative.

We build our visual cues out of rules. At the cost of a much bigger DVI file, this is to be preferred over using characters (1) because we cannot be sure of their availability and (2) because their dimensions are fixed.

As with ruled boxes, we use a (dimension) to specify the width of the ruled elements. This dimension defaults to:
\[\testrulewidth=\boxrulewidth\]

Because we prefer whole numbers for specifying the dimensions, we often use even multiples of \testrulewidth. A second variable is introduced because of the stretch components of (skips). At the cost of some accuracy we can make this stretch visible.
\[\visiblestretchtrue\]
\[\newdimen\testrulewidth \testrulewidth=\boxrulewidth\]
\[\newif\ifvisiblestretch \visiblestretchfalse\]

We start with the easiest part, the fills. The scheme we follow is visual filling – going back – normal filling. Visualizing is implemented using \cleaders. Because the (box) that follows this command is constructed only once, the \copy is not really a prerequisite. We prefer using a \normalhbox here instead of a \hbox.
\[\setvisiblehfilbox#1\to\hfil#2\hfil\hfil\hfil\hfil\hfil#4\]
\[\def\doruledhfiller#1#2#3#4\{#1#2\bgroup\dontinterfere\dontcomplain\setvisiblehfilbox0\to\hfil#3\hfil#4\copy0\copy2\bgroup\testrulewidth=\boxrulewidth\]

The horizontal fillers differ in their boundary visualization. Watch the small dots. Fillers can be combined within reasonable margins.

```
\hss………………………………………………………………………………………………test
\hfil………………………………………………………………………………………………test
\hfil\hfil…………………………………………………………………………………………test
\hfil\hfil\hfil…………………………………………………………………………………………test

The negative counterparts are visualized, but seldom become visible, apart from their boundaries.

\hfilneg………………………………………………………………………………………………test
\hfilneg\hfil…………………………………………………………………………………………test

Although leaders are used for visualizing, they are visualized themselves correctly as the next example shows.
```

All five substitutions use the same auxiliary macro. Watch the positive first – negative next approach.

```
\def\ruledhss% 35
213 \{\doruledhfiller\normalhss\normalhfilneg\}
214 \{0\{0\}
\def\ruledhfil% 36
215 \{\doruledhfiller\normalhfil\normalhfilneg\}
216 \{10\{-6\}
\def\ruledhfill% 37
217 \{\doruledhfiller\normalhfill\normalhfillneg\}
218 \{18\{-14\}
\def\ruledhfilneg% 38
219 \{\doruledhfiller\normalhfilneg\normalhfill\}
220 \{-6\{10\}
\def\ruledhfillneg% 39
221 \{\doruledhfiller\normalhfilneg\normalhfill\}
222 \{-14\{18\}

The vertical mode commands adopt the same visualization scheme, but are implemented in a slightly different way.

```
\def\setvisiblevfilbox#1\to#2#3#4% 40
223 \{\setbox#1=\normalvcue
224 \{\vrule
225 \{\!!width#2\testrulewidth
226 \{\!!height#3\testrulewidth
227 \{\!!depth#4\testrulewidth
228 \}
229 \smashbox#1\}
41
\def\doruledvfiller#1#2#3% 41
230 \{\setvisiblevfilbox\}
231 \{\to#2#3#4\}
232 \{\setbox#1=\normalvcue
233 \{\vrule
234 \{\!!width#2\testrulewidth
235 \{\!!height#3\testrulewidth
236 \{\!!depth#4\testrulewidth
237 \}
238 \smashbox#1\}
42
\def\dontrulefiller#1#2#3% 42
239 \{\setvisiblevfilbox\}
240 \{\to#2#3#4\}
241 \{\setbox#1=\normalvcue
242 \{\vrule
243 \{\!!width#2\testrulewidth
244 \{\!!height#3\testrulewidth
245 \{\!!depth#4\testrulewidth
246 \}
247 \smashbox#1\}
```

Because they act the same as their horizontal counterparts we only show a few examples.

\vss
\vfil
\vfill
\vfilneg
\vfillneg

Keep in mind that \vfillneg is not part of PLAIN TeX, but are mimicked by a macro.

Skips differ from kerns in two important aspects:
- line and pagebreaks are allowed at a skip
- skips can have a positive and/or negative stretch component

Stated a bit different: kerns are fixed skips at which no line or pagebreak can occur. Because skips have a more open character, they are visualized in a open way.

When skips have a stretch component, this is visualized by means of a dashed line. Positive skips are on top of the baseline, negative ones are below it. This way we can show the combined results. An alternative visualization of stretch could be drawing the mid line over a length of the stretch, in positive or negative direction.
We are less fortunate when implementing the vertical skips. This is a direct result of interference between the boxes that visualize the skip and skip removal at a pagebreak. Normally skips disappear at the top of a page, but not of course when visualized in a \vbox. A quite perfect simulation could have been built if we would have had available two more
primitives: \texttt{\tmpnop} and \texttt{\tmpvnop}. These new primitives could stand for boxes that are visible but are not taken into account in any way. They are there for us, but not for \TeX{}.

\begin{verbatim}
  first line \vskip +30pt plus 5pt
  second line \vskip +30pt
  third line \vskip -10pt plus 5pt
  fourth line \par
  fifth line \vskip 0pt
  sixth line
\end{verbatim}

We have to postpone \texttt{\prevdepth}. Although this precaution probably is not completely waterproof, it works quite well.

\begin{verbatim}
\def\dodoruledvskip{
  \nextdepth=\prevdepth
  \dontinterfere \dontcomplain \offinterlineskip \investigateskip \scratchskip
  \ifzero
    \setbox0=\normalvcue {
      \vrule \!!width32\testrulewidth \!!height2\testrulewidth \!!depth2\testrulewidth}
  \else
    \setbox0=\normalvbox to \ifnegative-\fi\scratchskip {
      \hrule \!!width16\testrulewidth \!!height2\testrulewidth \ifflexible
        \cleaders
        \normalhbox to 16\testrulewidth
        {\normalhss \normalvbox {
          \normalvskip 2\testrulewidth \hrule \!!width2\testrulewidth \!!height2\testrulewidth 
          \normalvskip 2\testrulewidth} \normalhss}
      \normalvfill \hrule \!!width16\testrulewidth \!!height2\testrulewidth}
    \setbox2=\normalvbox to \ht0 {
      \hrule \!!width2\testrulewidth \!!height\ht0}
    \ifnegative
      \ht0=\!!zeropoint
      \setbox0=\normalhbox {\normalhss}
    \fi
  \else
    \normalvfill
  \fi
  \smashbox0 \smashbox2
  \setbox0=\normalvcue {
    \box2\box0}
  \setbox0=\normalvbox {
    \ifnegative
      \normalvskip\scratchskip
    \fi
\end{verbatim}
We try to avoid interfering at the top of a page. Of course we only do so when we are in the main vertical list.
After having seen the horizontal ones, the vertical kerns will not surprise us. In this example we use \par to switch to vertical mode.

Like before, we have to postpone \prevdepth. If we leave out this trick, we got ourselves some wrong spacing.
The non-primitive glue commands are treated as kerns with stretch. This stretch is presented as a dashed line. I have to admit that until now, I’ve never used these glue commands.

\begin{verbatim}
\hglue +30pt plus 5pt one \hglue -10pt plus 5pt two \hglue 0pt three \hglue +30pt five
\end{verbatim}
Mathematical kerns and skips are specified in mu. This font related unit is incompatible with those of \emph{dimensions} and \emph{skips}. Because in math mode spacing is often a very subtle matter, we've used a very simple, not overloaded way to show them.
After presenting fills, skips, kerns and glue we’ve come to see penalties. In the first implementation — most of the time needed to develop this set of macros went into testing different types of visualization — penalties were mere small blocks with one black half, depending on the sign. This most recent version also gives an indication of the amount of penalty. Penalties can go from less than \(-10000\) to over \(+10000\), and their behavior is somewhat non-linear, with some values having special meanings. We therefore decided not to use its value for a linear indicator.

The small sticks at the side of the penalty indicate its size. The next example shows the positive and negative penalties of 0, 1, 10, 100, 1000 and 10000.

This way stacked penalties of different severance can be shown in combination.
The size of a vertical penalty is also shown on the horizontal axis. This way there is less interference with the often preceding or following skips and kerns.
For those who want to manipulate the visual cues in detail, we have grouped them.
All these nice options come together in two macros. The first one turns the options on, the second turns them off. Both macros only do their job when we are actually showing the composition.

```latex
\showingcompositiontrue
\showcomposition
```

Because the output routine can do tricky things, like multiple column typesetting and manipulation of the pagebody, shifting things around and so on, the macro `\dontshowcomposition` best can be called when we enter this routine. Too much visual cues just don’t make sense. In \LaTeX{} this has been taken care of.

```latex
\newif\ifshowingcomposition
\def\showcomposition{%
  \ifshowingcomposition
    \showfils
    \showboxes
    \showskips
    \showpenalties
  \fi
}
\def\dontshowcomposition{%
  \ifshowingcomposition
    \dontshowfils
    \dontshowboxes
    \dontshowskips
    \dontshowpenalties
  \fi
}
```

Just to make things even more easy, we have defined:

```latex
\showmakeup
```

For the sake of those who don’t (yet) use \LaTeX{} we preset `\defaulttestrulewidth` to the already set value. Otherwise we default to a corps related value.

```latex
\def\defaulttestrulewidth{.2pt}
```

Beware, it’s a macro not a \texttt{\langle dimension\rangle}.

```latex
\ife\korpsgrootte\undefined
  \edef\defaulttestrulewidth{\the\testrulewidth}
\else
  \def\defaulttestrulewidth{.02\korpsgrootte} % still dutch
\fi
```

```latex
\def\showmakeup{%
  \testrulewidth=\defaulttestrulewidth
  \showingcompositiontrue
  \showcomposition
}
```

The documented source you have been reading was processed using some surrogate makeup. When this file is processed in \LaTeX{}, a few more examples show up here, like a local table of contents and a local register.