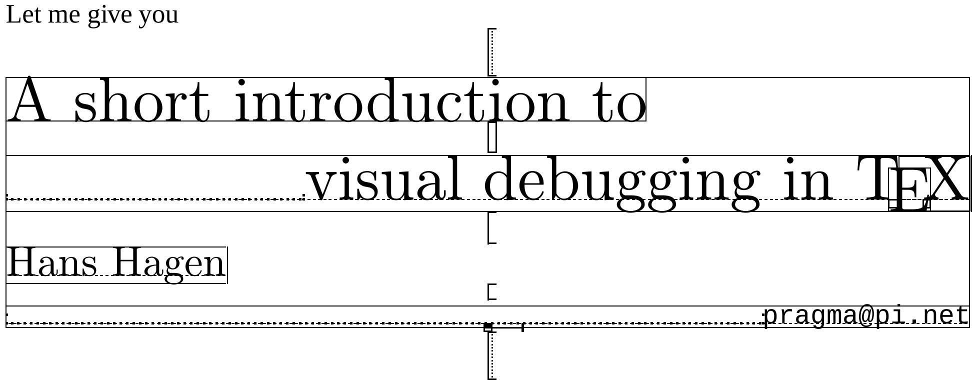
Visual Debugging in TEX

a short introduction

Hans Hagen

September 25 1996

Let me give you



A short introduction to
visual debugging in TEX
Hans Hagen
pragma@pi.net

This kind of fancy heading shows some dotted lines, rules and peculiar visual symbols. A more close observation learns that in fact it is some endoscopic view in what is often called TEX's stomach. For those readers who have planned to skip the rest of this article, here is how the magic is done:

```
\input supp-vis \showmakeup
```

For those who want to take a closer look at all those kerns, skips and penalties, this articles can be of some help. Although this kind of stuff often attracts the more hacking type of reader, the module described here can be of great help and provide a lot of fun to all TEX users, whatever macropackage they use.

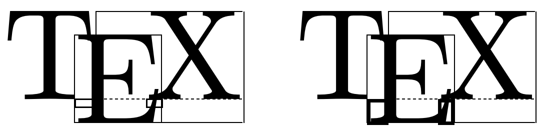
When TEX builds paragraphs and pages, it takes a lot into account. Even after years of writing macros the interference of skips, kerns, penalties, boxes and rules sometimes surprises me. One must always be aware of interline skips, top of page skips, good breaks and no breaks, either user supplied or system generated.

The idea to build some visualization macros was born while I was documenting the source of CONTEXT. Because this package is quite complete, the full documentation will be laid down in thousands of pages. Such technical documentation cannot go without showing how things are done. Because most macros at the user level have some visual impact, I decided to build a visualization tool. After having written this bunch of macros, their second purpose soon became visual debugging.

The concept is rather simple: replace the primitives `\.box`, `\.skip`, `\kern`, `\penalty`, `\.glue`, `\.ss`, `\.fil.` and `\.fil.neg` by macros that makes them visible. Most advanced TEX tutorials give examples of adapting the primitive `\par`, but somehow tampering with other TEX primitives is considered more tricky. Although the name primitive suggest that they are somehow fixed, even primitives can be `\let`'d or `\def`'d to something else. Temporary superseding the `\font` primitive is for instance needed when one wants to postpone loading of fonts in Plain TEX.

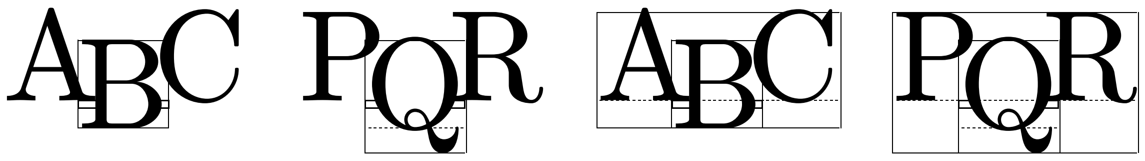
One can imagine that replacing `\hbox` with something else can have disastrous consequences. Primitives like `\setbox` expect a box and setting `\hbox` to `\relax` will surely lead to loud complaints. Some first experiments showed however that substitution was surprisingly easy. More time was spent on finding a sort of replacement that does not conflict visually when more primitives are given in a row.

Let's start with a well known piece of text. We've blown it up a bit, so we can see what happens.

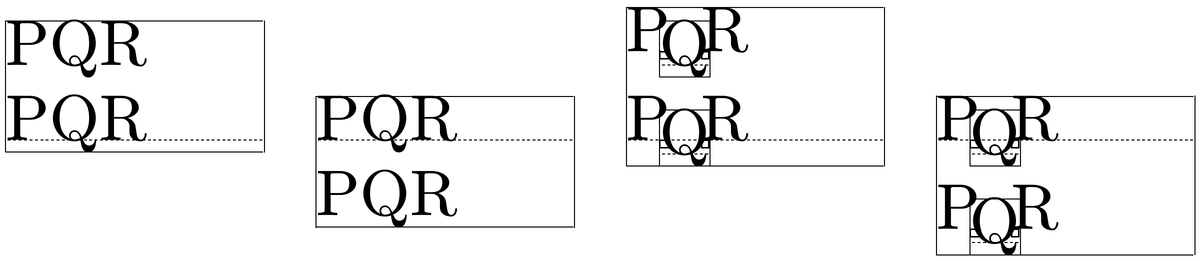


Here we see a T, followed by a kern, a boxed E, another kern and a X. The kerns have a negative sign and are visualized as small rectangles. Negative values are drawn left of their insertion point. The second TEX has exaggerated cues.

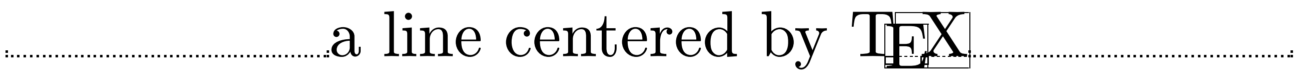
The three uppercase characters that make up TEX have no descenders. The next example shows a few more TEXed characters. This time we've got them boxed, so we can see what happens to the baseline of this combination of characters. Lowering the B and Q does not influence the baseline, which is what we expect.



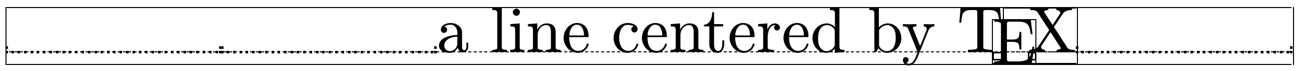
Vertical boxes come in two flavors. The default vertical box `\vbox` inherits it's baseline from the last line, while `\vtop` takes the baseline of the first line.



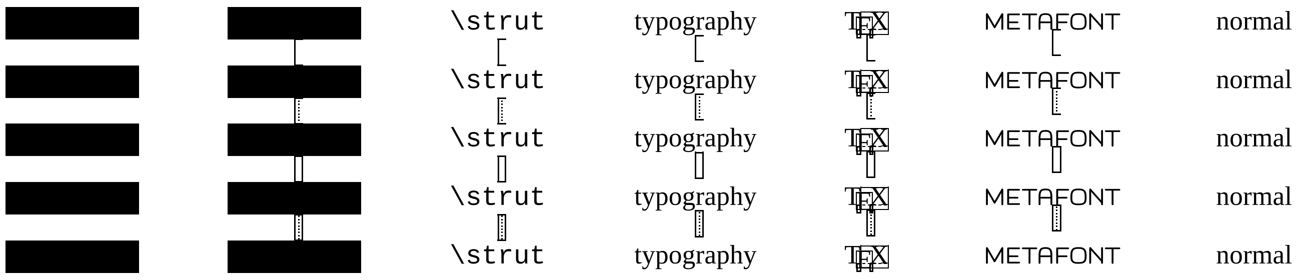
Visualization of fills is no problem either. In the centered line shown below the piece of text has some `\hfil`'s around it.



The same one, showing the surrounding box and two `\hfil`'s at the left of the text, looks like:

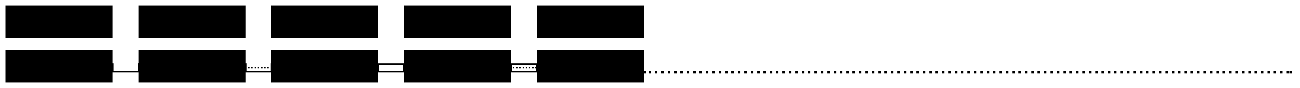


When using substitutes for the primitives mentioned, keeping the spacing intact is not always trivial. Especially the vertical spacing is very sensitive to interference. The next examples show us that at least normal situations can be handled well.

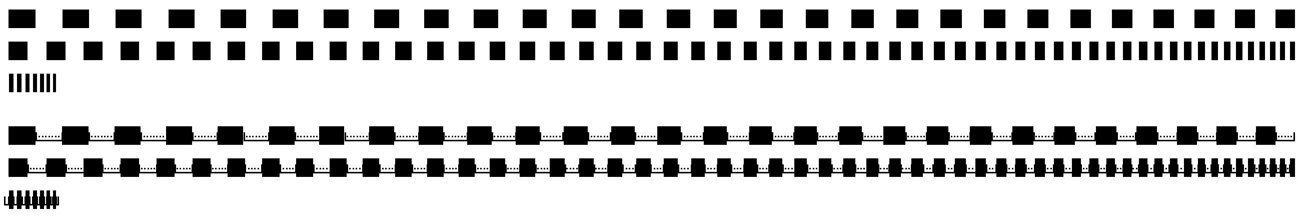


Here we see some positive vertical cues. Their negative counterparts are drawn left of the axis. Top down we see a skip, another skip with some stretch, a kern and some glue. A penalty of 100 looks like this and can be negative too. Skips, kerns and glue, which by the way is a Plain TEX macro and not a primitive, are shown at their natural size. Penalties are drawn in ranges, which are tuned to the most common cases. Combinations of penalties show up all right as we can see in where we have inserted penalties of 10000, 100 and 1.

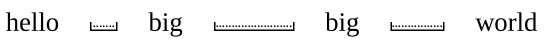
Horizontal spacing is less sensitive than vertical spacing. Here we don't have to take interline spacing and previous depths into account. Just to prove that things work, we show a similar example here. As a bonus we've added `\hss`.



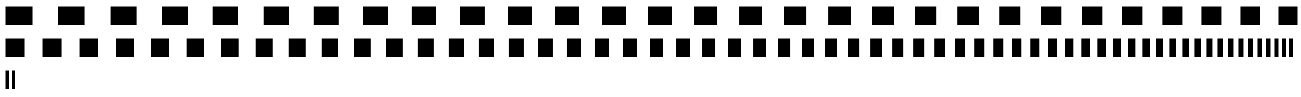
When we are typesetting in horizontal mode we have to preserve linebreaking. The next example shows a dummy paragraph with skips.

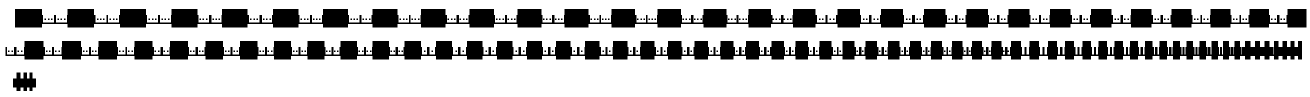


In this example it's hard to see that the stretch is equally distributed around the skip. The next line of text shows this feature in full glory. This feature is disabled by default.



Now look what happens when we combine two horizontal skips. This time TEX is not able to remove the visual cues. A similar situation occurs at a pagebreak. This kind of tricky situations can only be solved by an invisible kind of box, which is unfortunately not part of TEX. Of course we can backtrack skips, kerns and penalties, but such a, still not perfect, solution only complicates the macros beyond understanding.





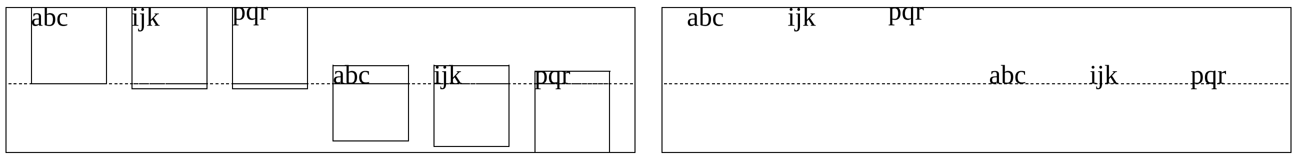
Mathematical spacing is implemented too, but due to the font-bound character, its visualization is the least impressive: $x y$ and $x y$ for math kern and math skip of 7 mu.

The next set of examples shows how vertical boxes are aligned when pasted together in a horizontal box. When I was messing around a bit with these samples, I became aware of some side effects that normally go unnoticed probably because they are quite natural. Confronted with these effects, I first thought that the visualization macros were somehow responsible, but additional testing proved otherwise. Of course one can never be sure, but rereading some paragraphs in Victor Eijkhouts TEX by Topic learned me that indeed such effects occur.

The samples are built up in the following way. Here the dots stand for some trailing text and/or macros.

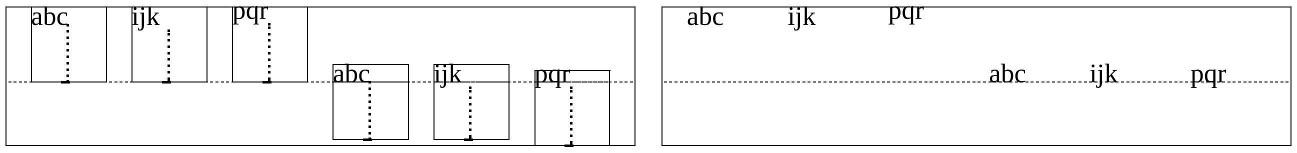
```
\hbox to \hsize
  {\hss
   \vbox to 1cm{\hsize.15\hsize abc\par ... }\hss
   \vbox to 1cm{\hsize.15\hsize ijk\par ... }\hss
   \vbox to 1cm{\hsize.15\hsize pqr\par ... }\hss
   \vtop to 1cm{\hsize.15\hsize abc\par ... }\hss
   \vtop to 1cm{\hsize.15\hsize ijk\par ... }\hss
   \vtop to 1cm{\hsize.15\hsize pqr\par ... }\hss}
```

We show both the visualized example and the natural one. The latter illustrates compatibility. When we insert nothing, this pack of boxes looks like:

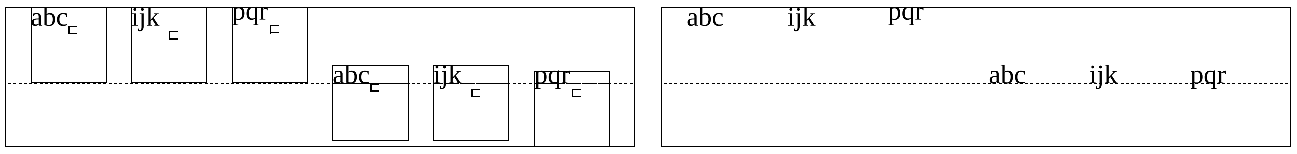


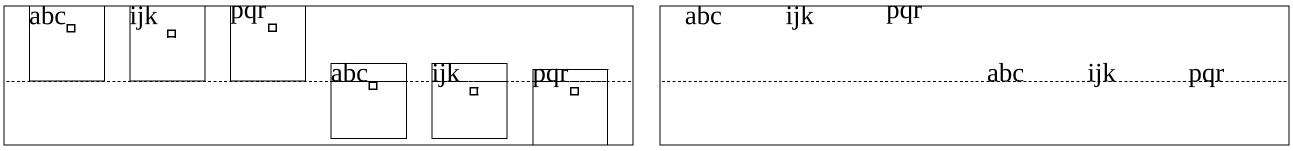
The first box has the height we expect. The second and third box also has the desired height, but here the depth of the j and q has migrated to the surrounding box. The height and depth of the fourth box totals to 1 cm, and we don't recognize the 1 cm in one of those dimensions. The last two boxes behave a bit unexpected. Here the depth is added to the height we specified. These last three situations learn us that specifying the height of a `\vtop` does not always make that much sense.

Now watch what happens when we add a `\vss`. This time the ijk and pqr boxes behave as expected and we end up with six boxes of 1 cm. Seeing is believing.

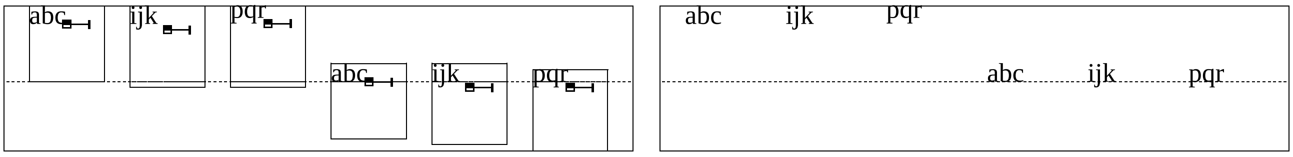


In most cases, one will add some kind of glue to a box, just to get rid of those underfull messages. It's good to be aware of the fact that adding glue does a bit more. Adding a `\vskip` or `\kern` has the same effect.

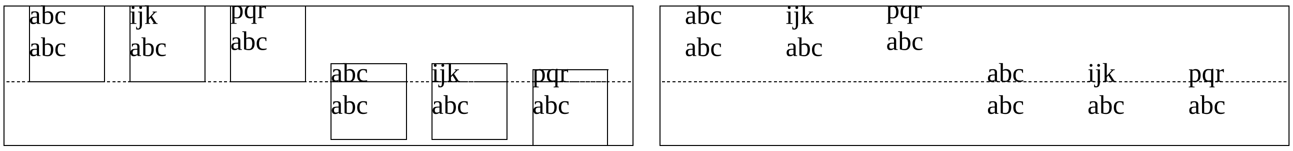


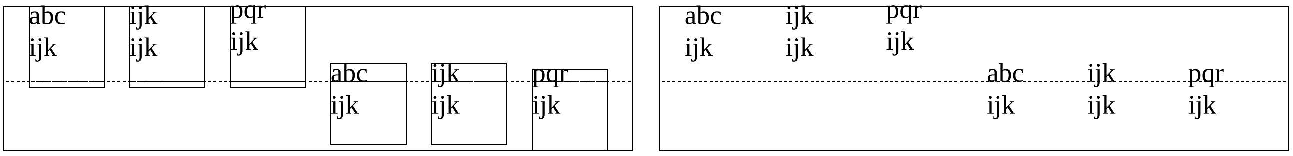


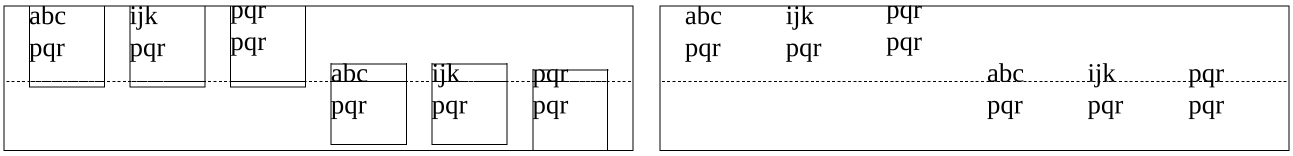
Adding a very large skip or kern makes no difference so we stick to these 3 pt examples. A penalty on the other hand has no effect. Here we get the same results as in the first example.



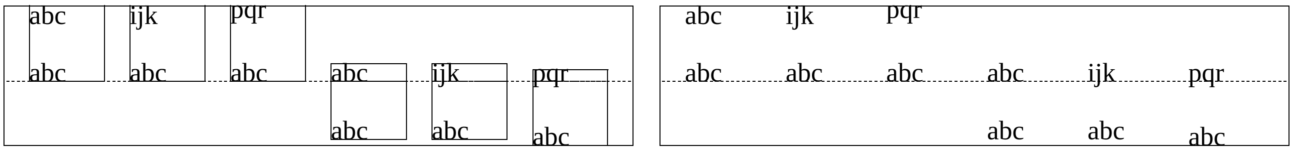
When we add some boxed text, the height and depth of the surrounding box depend on the depth of the (last) line. Here we show what happens when we insert a `\hbox`.

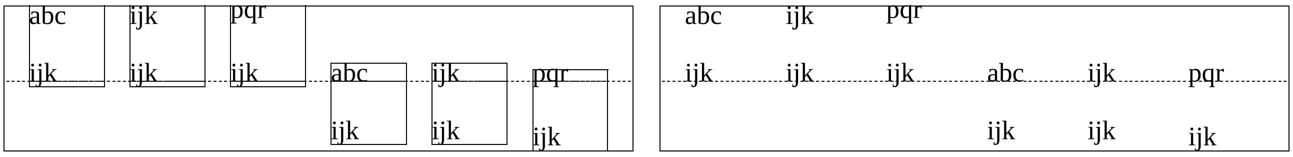


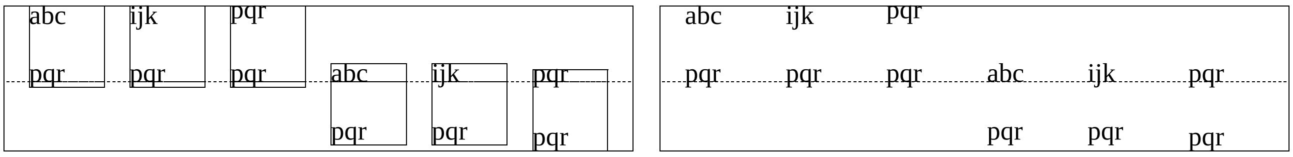




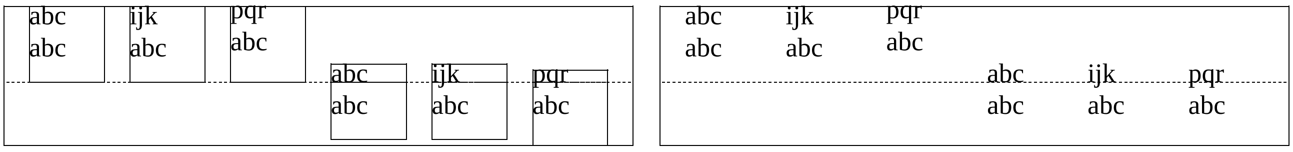
When we put the characters in a `\hbox` and `\unhbox` this box, we get different results. Just take a close look at the next set of boxes.

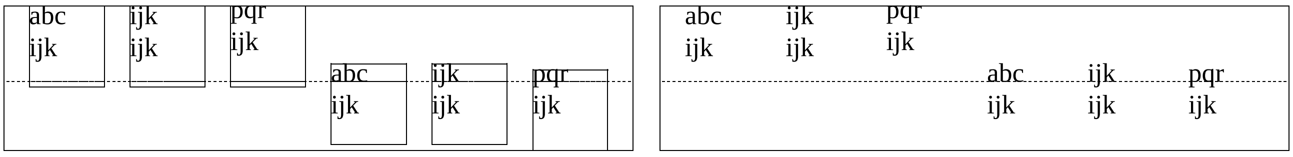


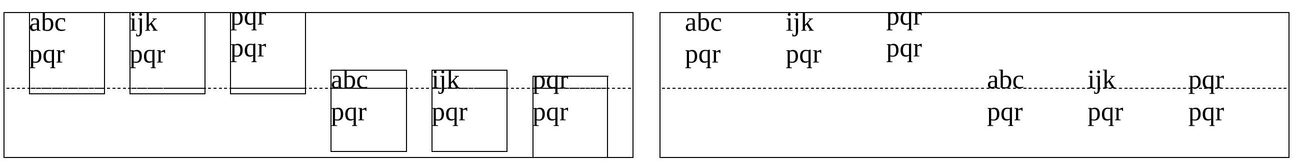




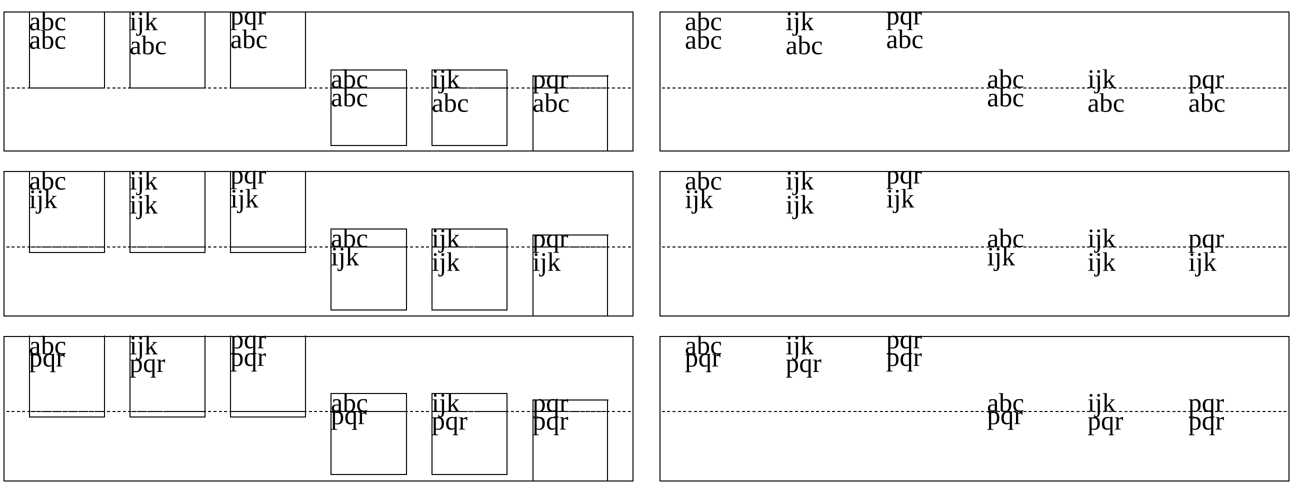
Things looks different when we add a `\vbox`. Just adding one looks like this:







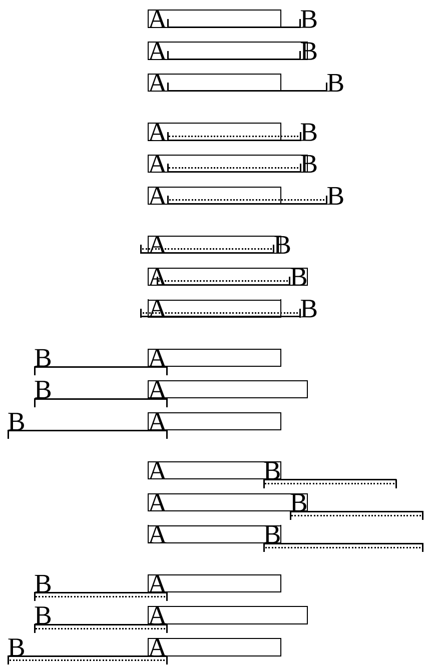
Adding an `\unvbox`'ed one looks a bit different. This kind of tests kan be both very confusing and instructive. It's a chalenge to deduce some systematic behavior from them.



I could show some more examples, like vertical boxes with more lines or `\vtop`'s. The examples shown here at least make clear that when we start manipulating boxes, we have to be aware of side effects.

Close reading of the TEXbook learns that the effects of the skip stretch components `plus` and `minus` sometimes depend on the context. Take a look at the next set of boxes.

```
\hbox to 5em A\hskip 5em B
\hbox to 6em A\hskip 5em B
\hbox to 5em A\hskip 6em B

\hbox to 5em A\hskip 5em plus 1em B
\hbox to 6em A\hskip 5em plus 1em B
\hbox to 5em A\hskip 6em plus 1em B

\hbox to 5em A\hskip 5em minus 1em B
\hbox to 6em A\hskip 5em minus 1em B
\hbox to 5em A\hskip 6em minus 1em B

\hbox to 5em A\hskip -5em B
\hbox to 6em A\hskip -5em B
\hbox to 5em A\hskip -6em B

\hbox to 5em A\hskip -5em plus 1em B
\hbox to 6em A\hskip -5em plus 1em B
\hbox to 5em A\hskip -6em plus 1em B

\hbox to 5em A\hskip -5em minus 1em B
\hbox to 6em A\hskip -5em minus 1em B
\hbox to 5em A\hskip -6em minus 1em B
```



Line- and pagebreaks can in no way be handled 100% perfect. TEX clears out redundant skips and penalties when crossing lines and pages. Making skips and penalties visible calls for the use of boxes and rules. A more perfect visualizer can be build when two more box primitives will be available: `\hnop` and `\vnop`. Both primitives should act like normal boxes when being manipulated, but should be kept out of paragraph and pagebreak calculation. They should be visible in the output but invisible for TEX itself. Lacking these primitives, visualization of sequences of skips and penalties will lead to non-compatible results.

Like the colored verbatim modules described in a previous article, the visual debugger module can be used on top of Plain TEX. Both modules only use a few general system macros, which are supplied in a small miscellaneous module. For CONTEXT users, visualization is always available, because it's just one of the standard features. For users of Plain TEX (or for those who use other packages) the next commands will do the trick:

```
\input supp-vis
```

When this module is loaded, the command `\showmakeup` will turn on the visualization. Users can turn on and off some features, like alignment of vertical cues, individual categories of cues and the visible baseline. The macros and features are explained in detail in the documented module itself.

The `supp` stands for general support. The symmetrical verbatim module, which supports typesetting of colored TEX sources that we presented in a previous article, belongs to this category too. When used outside CONTEXT, both modules automatically fall back on a small module `supp-mis`, which implements poor mans alternatives for a few system macros.

Visualization can best be used grouped. Depending on the number of primitives used, the output can be huge when one processes whole pages. Plain TEX's pagebody routine is both simple and effective. Unfortunately, the more flexibility one wants, the more complicated this routine becomes. In CONTEXT for instance this routine has to deal with multiple headers and footers, backgrounds, logos, multiple margins, interaction menus, navigational tools and a few more. Therefore we turn off visualization as long as we are building the page. The same goes for multi-column handling and some Plain TEX macros like `\llap` and `\rlap`.

In Plain TEX it's not that hard to turn things off temporary. Just give the next code a try:

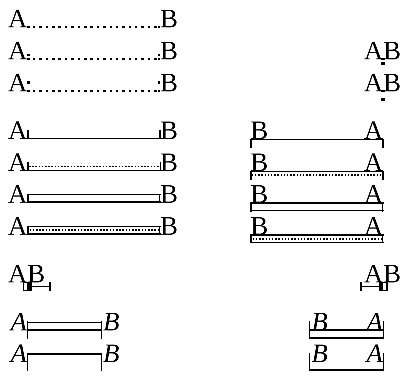
```
\input supp-vis  \output{\dontshowcomposition\plainoutput}
\showmakeup      \hbox{so much} \eject \hbox{for now} \end
```

In CONTEXT there are some more similar facilities, like general layout, `\strut` and baseline visualization. At the moment, the functionality of this module is limited to the primitives mentioned. We already visualize the mathematical skips, but when needed, we will extend this module with some useful math debugging facilities. A year from now, this module probably will be a bit more advanced anyway.

I could show some more instructive examples, but for producing those, I have to depend a bit too much on CONTEXT for processing. For the same reason the next article, which describes the module itself, lacks some useful functionality.

Let's summarize the cues. Positive horizontal cues are drawn on top of and negative ones under the baseline. The negative cues are drawn in the negative direction. Vertical cues are drawn left or right of the current point (or halfway the `\hsize`) and they too honor the direction. In the next table we only show the horizontal cues.

```
\hss
\hfil
\hfill

\hskip 5em
\hskip 5em plus 1em
\kern 5em
\hglue 5em plus 1em

\penalty 200

\mskip 50mu plus 1mu
\mkern 50mu
```



Kerns and penalties are treated according to the current mode, which is horizontal or vertical. Zero cues are a special case. A zero horizontal skip for instance shows up as ❘ , a kern look like ▯ and a zero penalty becomes ▫ . As far as possible, different kind of cues add up nicely.