low level

TeX

balancing

Contents

1 Introduction

This is work in progress as per end 2024 these mechanisms are still in flux. We expect them to be stable around the ConTEXt meeting in 2025. The text is not corrected, so feel free to comment.

This manual is about a new (sort of fundamental) feature that got added to LuaMetaTEX when we started upgrading column sets. In TEX we have a par builder that does a multi-pass optimization where it considers various solutions based on tolerance, penalties, demerits etc. The page builder on the other hand is forward looking and backtracks to a previous break when there is an overflow. The balancing mechanism discussed here is basically a page builder operating like the par builder: it looks at the whole picture.

In order to make this a useful mechanism the engine also permits intercepting the main vertical list, so we start by introducing this.

2 Intercepting the MVL

When content gets processed it's added to a list. We can be in horizontal mode or vertical mode (let's forget about math mode). In vertical mode we can be in a box context (say `\vbox`) or in what is called the main vertical list: the one that makes the page. But what is page? When TEX has collected enough to match the criteria set by `\pagegoal` which starts out as `\vsize`, it will call the so called output routine which basically is expanding the `\output` token list. That routine had do so something with the box that has the collected material. It can become a page, likely with the content wrapped in a page body with headers and footers and such, but it can also be stored for later assembly, for instance in multiple columns, or after some analysis fed back into the main vertical list.

For various mechanisms it matters if they are used inside a contained boxed environment or in the more liberal main vertical list (from now on called mvl). That's why we can intercept the mvl and use it later. Intercepting works as follows:

```
\beginmvl 1
various content
\endmvl

\beginmvl 2
various content
\endmvl
```

When at some point you want this content, you can do this:

```
\setbox\scratchboxone\flushmvl 2
\setbox\scratchboxtwo\flushmvl 1
```

and then do whatever is needed. You can see what goes on with:

```
\tracingmvl 1
```

There is not much more to say other than that this is the way to operate on content as if it were added to the page which can be different from collecting something in a vertical box. Think of various callbacks that can differ for the mvl and a box.

The `\beginmvl` primitive takes a number or a set of keywords, as in:

```
\beginmvl
    index   1
    options \numexpr "01 + "04\relax
\relax
```

There is of course some possible interference with mechanism that check the page properties like `\pagegoal`. If needed one can check this:

```
\ifcase\mvlcurrentlyactive
  % main mvl
\or
  % first one
\else
  % other ones
\fi
```

Possible applications of this mechanism are the mentioned columns and parallel, independent, streams. However for that we need to be able to manipulate the collected content. Actually, the next manipulator preceded the capturing, because we first wanted to make sure that what we had in mind made sense.

The `beginmvl` also accepts keywords. You can specify an `index` (an integer), a `prevdepth` (dimensions) and `options` (an integer bitset). Possible option bit related values are:

`0x1`	ignore prevdepth	`\ignoreprevdepthmvloptioncode`
`0x2`	no prevdepth	`\noprevdepthmvloptioncode`
`0x4`	discard top	`\discardtopmvloptioncode`
`0x8`	discard bottom	`\discardbottommvloptioncode`

Here the last column is a numeric alias available in ConTEXt. More options are likely to show up. When we eventually will balance these lists the routine will deal with the discardables (like glue) but one can also remove them via the options.

```
\beginmvl
    index     1
    prevdepth 0pt
    options  \discardtopmvloptioncode
\relax
\scratchdimen\prevdepth
\dontleavehmode
\quad\the\mvlcurrentlyactive\quad\the\scratchdimen
\quad\blackrule[height=\strutht,depth=\strutdp,color=darkred]
\endmvl

\ruledhbox {\llap{1\quad}\flushmvl 1}
```

1   1 0.0pt █

```
\beginmvl
    index  2
    options \numexpr
                \ignoreprevdepthmvloptioncode
              + \discardtopmvloptioncode
            \relax
\relax
\scratchdimen\prevdepth
\dontleavehmode
\quad\the\mvlcurrentlyactive\quad\the\scratchdimen
\quad\blackrule[height=\strutht,depth=\strutdp,color=darkred]
```

```
\endmvl

\ruledhbox {\llap{2\quad}\flushmvl 2}
```

```
2    2  -1000.0pt  ■
```

```
\beginmvl 3 % when no keywords are used we expect a number
\scratchdimen\prevdepth
\dontleavehmode
\quad\the\mvlcurrentlyactive\quad\the\scratchdimen
\quad\blackrule[height=\strutht,depth=\strutdp,color=darkred]
\endmvl

\ruledhbox {\llap{3\quad}\flushmvl 3}
```

```
3    3  0.0pt  ■
```

```
\beginmvl index 4 options 1
\scratchdimen\prevdepth
\dontleavehmode
\quad\the\mvlcurrentlyactive\quad\the\scratchdimen
\quad\blackrule[height=\strutht,depth=\strutdp,color=darkred]
\endmvl

\ruledhbox {\llap{4\quad}\flushmvl 4}
```

```
4    4  -1000.0pt  ■
```

3 Balancing

Balancing is not referring to balancing columns but to ‘a result that looks well balanced’. Just like we want lines in a paragraph to look consistent with each other, something that is reflected in the (adjacent) demerits, we want the same with vertical split of pieces. For this purpose we took elements of the par builders to construct a (page) snippet builder. Here are some highlights:

- Instead of a pretolerance, tolerance and emergency pass we only enable the last two. In the par builder the pretolerance pass is the one without hyphenation.

- We seriously considered vertical discretionaries but eventually rejected the idea: we just don't expect users to go through the trouble of adding lots of split related pre, post and replace content. It's not hard to support it but in the end it also interfered with other demands that we had. We kept the code around for a while but then removed it. To mention one complication: if we add some new node we also need to intercept it in various callbacks that we already have in place in ConTEXt. As with horizontal discretionaries, we then need to go into the components and sometimes even need to make decisions what can not yet be made.

- As with the par builder, TEX will happily produce an overfull box when no solution is possible that fits the constraints. In a paragraph there are plenty spaces (with stretch) and discretionaries (with components that vary in width) which enlarges the solution space. In vertical material there is less possible so there an emergency pass really makes sense: better be underful than overful.
- In many cases there is no stretch available. There are also widow, club, shape and orphan penalties that can limit the solution space.
- When we look at splitting pages (and boxes) we see (split) top skip kick in. This is something that we need to provide one way ot the other. And as we have to do that, we can as well provide support for bottom skip. A horizontal analogue is protrusion, something that also has to be taken into account in a rather dynamic way, at the beginning or end of the currently analyzed line.
- There is no equivalent of hanging indentation but a shape makes sense. Here the shape defines heights, top and bottom skips and maybe more in the future. For that reason we use a keyword driven shape.
- Because we have so called par passes, it made sense to have something similar for balancing. This gives is the opportunity to experiment with various variables that drive the process.
- For those who read what we wrote about the par builder, it will not come as surprise that we also added extensive tracing and a callback for intercepting the results. This makes it possible to show the same detailed output as we can do for par passes.

It's about time for some examples but before we come to that it is good to roughly explain how the page builder works. When the page builder is triggered it will take elements from the contributions list and add them to the page. When doing that it keeps track of the height and depth as contributed by boxes and rules. Because it will discard glue and kerns it does some checking there. An important feature is that the depth is added in a next iteration. The routine also needs to look at inserts. The variables `\pagegoal` (original `\vsize` minus accumulated insert heights) and `\pagetotal` are compared and when we run over the target height the accumulated stretch and shrink in glue (when present) will be used to determine how bad this break is. If it is too bad, the previous best break will be taken. Penalties can make a possible break more or less attractive. When the output routine gets a split of page, the total is not reliable because we can have backtracked to the previous break. In LuaMetaTEX we have some more variables, like `\pagelastheight`, that give a better estimate of what we got.

In order to make the first lines align properly relative to the top of the page there is a variable `\topskip`. The height of the first line is at least that amount. The correction is calculated when the first contribution happens: a box or rule.

When we look at the balancer it is good to keep in mind that where the page builder stepwise adds and checks, the balancer looks at the whole picture. The page builder does a decent job but is less sophisticated than the par builder. There is a badness calculation, penalties are looked at, glue is taken into account but there are no demerits.

We want the balancer to work well with column sets that are very much grid based. But in getting there we had some hurdles to take. Because the algorithm (like the par builder) happily results in overfull boxes unless emergency stretch is set, pages can overflow. When there is no stretch and/or shrink using emergency stretch can give an underfull page.

The way out of this is to have non destructive trial passes and decrease the number of lines. Of course we can get short pages but when for instance it concerns a section title that gets moved this is no big deal. In a similar fashion splitting a multi-line formula is also okay.

- Collect the content in an mvl list and after that's done put the result in a box.
- Set up a balance shape that specifies the slots in in columns (normally a column is just a blob of text).
- Perform a trial balance run. As soon as an overfull page is seen, adapt the balance shape and do a new trial run.
- When we're fine, either because we reached the end without overfull column or by passing the set deadcycles value, quit the trial process and balance the original list using the most recent balance shape.
- Flush the result by fetching the topmost from the result split collection and feed it into the page flow. The boxed pseudo page will happily trigger the output routine that in turn construct the final page.

At some point we decided to support multiple mvl streams and therefore changed the last mentioned step. Because we store the whole column set we can as well also store the assembled page bodies. This way we can flush different streams into the same result.

- Flush the result by fetching the topmost from the result split collection and feed it into the page flow. Do this for every saved (mvl) stream.

- When we're done, the boxed pseudo pages will be flushed as pages. In the process, for every page we identify marks.

We are now ready to look at some examples. Here we also show what balance shapes do. These basically describe a sequence of slots to be filled. The last specification is used when we exceed the number of defined slots. These are just examples of simple situations, for real applications more code is needed.

We start with some content in a box. This can of course be a flushed mvl but here we just set it directly:

```
\setbox\scratchboxone\vbox\bgroup
    \hsize.30\hsize
    \samplefile{tufte}
\egroup
```

We will split this box in columns. If you are familiar with TEX you might know that a paragraph of text can follow a shape defined by `\parshape`. In a similar way as lines are split by width, we can split a vertical list by height. For that we define a balance shape:

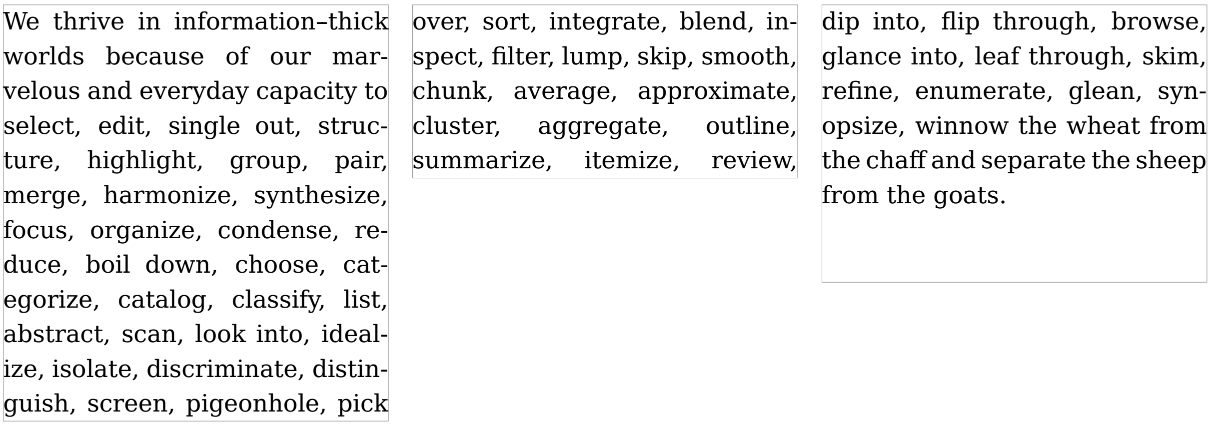
```
\balanceshape 3
    vsize      12\lineheight
    topskip    \strutht
    bottomskip \strutdp
next
    vsize       5\lineheight
    topskip    \strutht
    bottomskip \strutdp
next
    vsize      8\lineheight
    topskip    \strutht
    bottomskip \strutdp
\relax
```

```
\setbox\scratchboxtwo\vbalance\scratchboxone
```

Contrary to a `\parshape`, a `\balanceshape` is not wiped after the work is done. It also expects keys and values. As with `\parpasses` each step is separated by `next`. This makes it an extensible mechanism. Finally we will split the box according to this shape:

```
\hbox \bgroup
    \localcontrolledendless {%
        \ifvoid\scratchboxtwo
            \expandafter\quitloop
        \else
            \setbox\scratchbox\ruledhbox\bgroup
                \vbalancedbox\scratchboxtwo
            \egroup
            \vbox to 12\lineheight \bgroup
                \box\scratchbox
                \vfill
            \egroup
            \hskip1em
        \fi
    }\unskip
\egroup
```

The result is shown here:



We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick

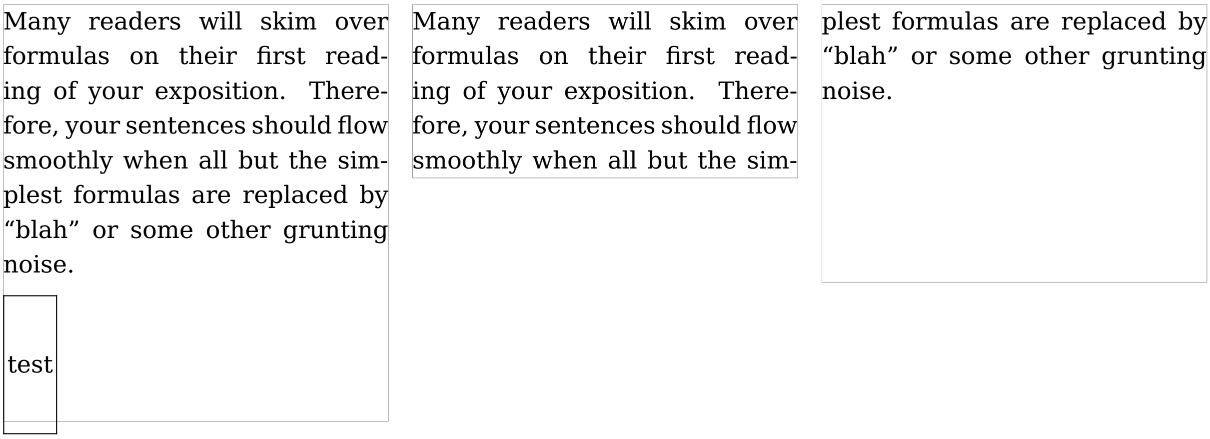
over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review,

dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

Like the par builder we can end up with overfull boxes but we can deal with that by using trial runs.

```
\setbox\scratchboxtwo\vbalance\scratchboxone trial
```

In that case the result is made from empty boxes so the original is not disturbed. Here we show an overflow, so in the first resulting box you can compare the height with the requested one and when it's larger you can decide to decrease the first height in the shape and try again.



Many readers will skim over formulas on their first reading of your exposition. Therefore, your sentences should flow smoothly when all but the simplest formulas are replaced by “blah” or some other grunting noise.

test

Many readers will skim over formulas on their first reading of your exposition. Therefore, your sentences should flow smoothly when all but the sim-

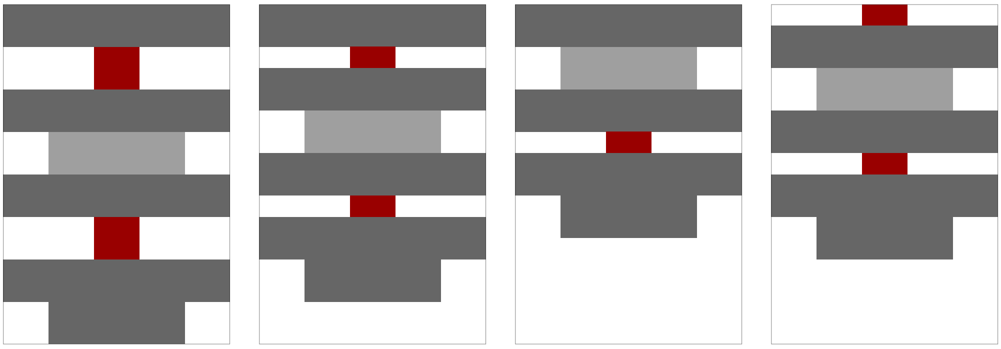
plest formulas are replaced by “blah” or some other grunting noise.

Of course that involves some juggling of the shape but after all we have Lua at our disposal so in the end it's all quite doable.

	real	**target**
1	167.8961pt	156.95874pt
2	65.39948pt	65.39948pt
3	49.17705pt	104.63916pt

Because the balancer can produce what otherwise the page builder produces, we need to handle the equivalent of top skip which is what the already shown `top` keyword takes care of. This means that the current slice (think current line in the par builder) has to take that into account. This can be compared to the left- and right protrusion in the par builder. When we typeset on a grid we have an additional demand.

When we surround (for instance a formula) with halfline spacing, we eventually have to return on the grid. One complication is that when we are in grid mode and use half line vertical spacing, we can end up in a situation where the initial half line space is on a previous page. That means that we need to use a larger top skip. This is not something that we want to burden the balancer with but we have ways to trick it into taking that compensation into account.



However, when we split in the middle of that segment, we can end up with a half line skip in a next slot because TEX will remove glue at the edge. So we end up with what we see in the third sequence above. We deal with that in a somewhat special way: a box as a discardable field which value will be taken into account as additional top value. That field is set and reset by glue options `0x20` and `0x40` that can be manipulated in Lua as part of some spacing model. Here we suffice by mentioning that it makes sure that (as in the fourth blob above) at the top we have a half line spacing.

4 Forcing breaks

Because the initial application of balancing was in column sets, we also need the ability to goto a next slot (step in a shape), column (possibly more steps), page (depending on the page state), and spread (for instance if we are doubles ided). For this we use `\balanceboundary`. It takes two values and when the boundary node triggers a callback in the builder these are passed along with a shape identifier and current shape slot. That callback can then signal back that we need to try a break here with a given penalty. Assuming that at the Lua end we know at which slot we have a slot, column, page or spread break. Multiple slots can be skipped by multiple boundaries. There is one pitfall: we need something in a slot in order to break at all, so one ends up with for instance:

```
\balanceboundary 3 1\relax
\vskip\zeropoint
\balanceboundary 3 0\relax
\vskip\zeropoint
\balanceboundary 3 0\relax
```

Here the `3` is just some value that the callback can use to determine its action (like goto a next page) and the second value provides a detail. Of course all depends on the intended usage. By using a callback we can force breaks while not burdening the engine with some hard coded solution. For example, in ConTEXt we used these (the values are these from experiments and might change:

first	**second**	**action**	**user interface**
1	1 or 0	goto next spread (1 initial, 0 follow up)	`\page[spread]`
2	1 or 0	goto next page (idem)	`\page`
3	1 or 0	goto next column (idem)	`\column`
4	1 or 0	goto next slot (idem)	`\column[slot]`
5	n	next slot when more than n lines	`\testroom[5]`
6	s	next slot when more than s scaled points	`\testroom[80pt]`

5 Marks

It is possible to synchronize the marks with those in the results of balanced segments with a few Lua helpers that do the same as the page builder does at the start of a page, while packaging the page and when wrapping it up. So, instead of split marks we can have real marks.

6 Inserts

Before we go into detail, we want to point out that when implementing a (balancing) mechanism as introduced above, decisions have to be made. In traditional TEX there is for instance an approach to inserts that involves splitting them over pages. In our case that is a bit harder to do but there are ways to deal with it. When deciding on an approach it helps that we know a bit what situations occur and where we can put some constraints. One can argue that solutions should be very generic because (for instance) a publisher has some specific demands but in practice those are not our audience. In decades of developing LuaTEX and LuaMetaTEX it's (ConTEXt) user demands and challenges that drives what gets implemented. Publishers, their suppliers, and large scale (commercial) users are pretty silent when it comes to development (and supporting it) while users communicate via meetings and mailing lists. Also, rendering of documents that have notes are often typeset kind of traditional.

Users on the other hand have come up with demands for columns, typesetting on the grid, multiple notes, balancing, and parallel content streams. The picture we get from that makes us confident that what we provide is generally enough and as users understand the issues at hand (maybe as side effect of struggling with solutions) it's not that hard to explain why constraints are in place. It makes more sense to have a limited reliable mechanism that deals with the kind of (foot)notes that known users need than to cook up some complex mechanism that caters potential specific demands by potential users. Of course we have our own challenges to deal with, even if the resulting features will probably not be used that often. So here are the criteria that make sense:

- We can assume a reasonable amount of notes.
- These are normally small with no (vertical) whitespace.
- Notes taking multiple lines may split.
- But we need to obey widow and club penalties.
- There can be math formulas but mostly inline.
- We need to keep them close to where they are referred from.

But,

- We can ignore complex conflicting demands.
- As long as we get some result, we're fine.
- So users have to check what comes out.
- We don't assume fully automated unattended usage.

And of course:

- Performance should be acceptable.
- User interfaces should be intuitive.
- Memory consumption should be reasonable.

We have users who use multiple note classes so that also has to be handled but again we don't need to come up with solutions that solve all possible demands. We can assume that when a book is published that needs them, the author will operate within the constraints.

We mentioned footnotes being handled by the page builder so how about them in these balanced slots? Given the above remarks, we assume sane usage, so for instance columns that have a single slot with possibly fixed content at the top or bottom (and maybe as part of the stream). The balancer handles notes by taking their height into account and when a result is used one can request the embedded inserts and deal with them. Again this is very macro package dependent. Among the features dealt with are space above and between a set of notes, which means that we need to identify the first and successive notes in a class. Given how the routine works, this is a dynamic feature of a line: the amount of space needed depends on how many inserts are within a slot. When we did some extreme tests with several classes of notes and multiple per column we saw runtime increasing because instead of a few passes we got a few hundred. In an extreme case of 800 passes to balance the result we noticed over four million checks for note related spacing. We could bring that down to one tenth so in the end we are still slower but less noticeable. Here are the helper primitives for inserts:

```
<state> = \boxinserts <box>
<box>   = \vbalancedinsert <box> <class>
<state> = \boxinserts <box>
```

A (foot)note implementation is very macro package dependent so the next example is just that: an example of using the available primitive. We start by populating a mvl with a sample text and a single footnote.

```
\begingroup
    \forgetall
    \beginmvl
```

```
        index 5
        options \numexpr
            \ignoreprevdepthmvloptioncode
          + \discardtopmvloptioncode
        \relax
    \relax
        \hsize .4tw
        Line 1 \par Line 2 \footnote {Note 1} \par Line 3 \par
        Line 4 \footnote {Note 2} \par Line 5 \par Line 6 \par
    \endmvl
\endgroup
```

We fetch the footnote number, which is one of many possible defined inserts

```
\cdef\currentnote{footnote}%
\scratchcounter\currentnoteinsertionnumber
```

The quick and dirty balancer uses a simple shape of 5 lines with normal strut properties. From the balanced result we take two columns. We test if there is an insert and take action when there is. Here we just filter the footnotes but there can of course be more. We overlay these notes over (under) the column that has them. So we work per column.

```
\begingroup
    \setbox\scratchboxone\flushmvl 5
    \balanceshape 1
        vsize      5lh
        topskip    1sh
        bottomskip 1sd
    \relax
    \setbox\scratchboxtwo\vbalance\scratchboxone
    \ruledhbox \bgroup
        \localcontrolledrepeat 2 {
          \ifnum\currentloopiterator > 1
            \hskip2\emwidth
          \fi
          \setbox\scratchboxthree\vbalancedbox\scratchboxtwo \relax
          \ifnum\boxinserts\scratchboxthree > 3
            \setbox\scratchboxfour\vbalancedinsert
                \scratchboxthree\scratchcounter
            \wd\scratchboxfour 0pt
            \box\scratchboxfour
```

```
          \fi
          \box\scratchboxthree
        }\unskip
    \egroup
\endgroup
```

The result is:

Line 1	Line 4[2]
Line 2[1]	Line 5
Line 3	Line 6
[1] Note 1	[2] Note 2

As we progressed we realized that the ‘balancer’ used in column sets can also be used for single columns and we can even support a mix of single and multi columns. There is however a problem: within a mvl we can deal with spacing but we can't do that reliable across mvl's and especially when we cross a page it becomes hard to identify if some (vertical) spacing is needed; we don't want it at the bottom or top of a page. This feature is too experimental to be discussed right now.

We assumed reasonable notes to be used but even if a user tries to keep notes small and avoid too many, there are cases where they might look like a paragraph and when there are more in a row, it might be that a column overflows. This is why we have some support for split notes. This is accomplished by two additional commands:

```
\setbox\scratchboxone\vbalance\scratchboxone\relax
\vbalanceddeinsert\scratchboxone\relax
```

Here we convert inserts in such a way that they are taken into account by the balancer so that multi-slot optimization takes place. Afterwards, when we loop over the result we can reconstruct the inserts:

```
\setbox\scratchboxtwo\vbalancedbox\scratchboxone
\vbalancedreinsert\scratchboxtwo\relax
```

Among the reasons that these are explicit actions, is that we want to experiment but also be able to see the effect by selectively enabling it. You can get better results by forcing depth correction.

```
\setbox\scratchboxone\vbalance\scratchboxone
\vbalanceddeinsert\scratchboxone forcedepth\relax
```

This will use the depth as defined by `\insertlinedepth` which is an insert class specific parameter, but discussing details of inserts is not what we do here. The reason for using a `\relax` in the above examples is that we want to stress that when keywords are involved, you need to prevent look-ahead, especially when an `\if...` or expandable loop follows, which is not uncommon when we balance.

It is possible to define top and bottom inserts but of course these need to be filtered and placed at the TeX end, so this is macro package specific. Here we just mention that it is possible to set `\insertstretch` and `\insertshrink` which will be taken into account. However, this can result in overlap so if indeed stretch or shrink is applied, the `handle_uinsert` callback should be used for bringing what actually gets inserted to the right dimensions. For now we consider this an experimental feature.

7 Discardables

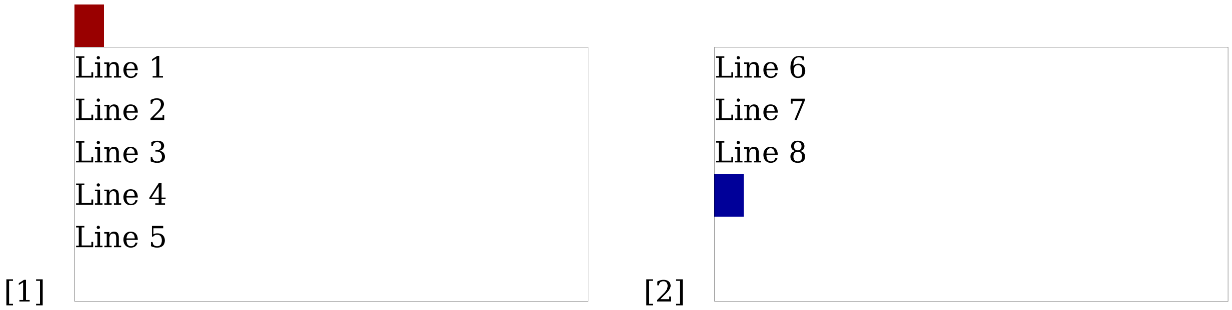
This is a preliminary explanation.

```
\begingroup
    \beginmvl
        index 5
        options \numexpr
            \ignoreprevdepthmvloptioncode
          + \discardtopmvloptioncode
        \relax
    \relax
        \hsize .4tw
        \par
        \vskip0pt
        {\darkred \hrule discardable height 1sh depth 1sd width 1em}
        \par
        % we need the strut because the rule obscures it .. todo
        \dorecurse{8}{\strut Line #1 \par}
        \vskip\zeropoint
        {\darkblue \hrule discardable height 1sh depth 1sd width 1em}
        \par
    \endmvl
\endgroup
```

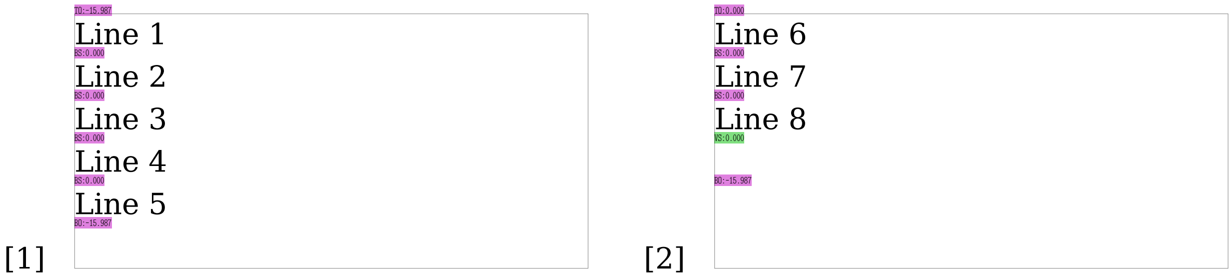
```
\setbox\scratchboxone\flushmvl 5
\balanceshape 1
    vsize       5lh
```

```
    topskip     1sh % see comment above
    bottomskip  1sd
    options     3
\relax
\setbox\scratchboxtwo\vbalance\scratchboxone\relax % lookhead

\hpack \bgroup
    \localcontrolledrepeat 3 {
        \ifvoid\scratchboxtwo\else
            \setbox\scratchboxthree\vbalancedbox\scratchboxtwo
            \ifvoid\scratchboxthree\else
                \dontleavehmode\llap{[\the\currentloopiterator]\quad}%
                \ruledhpack{\box\scratchboxthree}\par
            \fi
            \hskip 4em
        \fi
    }\unskip
\egroup
```

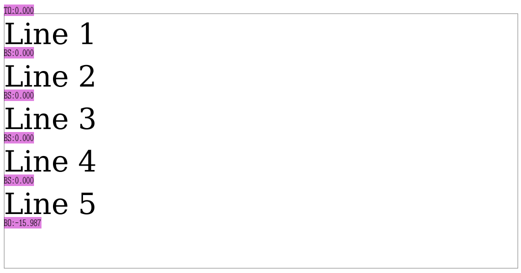


When at the top, the rule will be ignored and basically sticks out. When at the bottom the rule might end up in a zero dimension box. With `\vbalanceddiscard\scratchboxtwo` they will become an `\nohrule`. Basically we're talking of optional content. The `options` bitset in the shape definition tells if we have a top (1) and/ or bottom (2), here we have both (3) but in for instance column sets it depends.

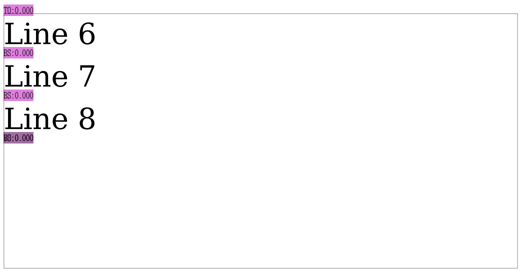


Here we actually still have the rule but marked as invisible. So, topskip has a negative amount. In the next case the `remove` keyword makes the rule go away in which case we also adapt the topskip accordingly.

[1]



[2]



You need to juggle a bit with skips and penalties to get this working as you like. Instead of rules you can also use boxes, for example before:

```
\vskip\zeropoint
\ruledvbox discardable {\hpack{\strut BEFORE}}
\par
```

and after:

```
\forgetall \par \vskip\zeropoint
\ruledvbox discardable {\hpack{\strut AFTER}}%
\penalty\minusone % !
\par
```

It currently is a playground so it might (and probably will) evolve. Although it was also made for a specific issue it might have other usage.

8 Passes

todo

```
\showmakeup[vpenalty,line]
\balancefinalpenalties 6 10000 9000 8000 7000 6000 5000\relax
\balancevsize 5\lineheight
\setbox\scratchbox\vbox{\dorecurse{1}{\samplefile{tufte}\footnote{!}\par}}
\vbalance\scratchbox
```

9 Passes

In LuaMetaTEX the par builder has been extended with additional features (like orphan, toddler and twin control) and the ability to define and apply multiple passes over the paragraph to get the best result. The balancer has a similar feature: `\balancepasses`. As with `\parpasses` we have an infrastructure for tracing.

```
% threshold
% tolerance
% looseness
% adjdemerits
% originalstretch
% emergencystretch
% emergencyfactor
% emergencypercentage
```

9 Colofon

Author	Hans Hagen
ConTEXt	2025.01.08 16:06
LuaMetaTEX	2.11.06 | 20250115
Support	www.pragma-ade.com contextgarden.net ntg-context@ntg.nl