Updates to “Automatically removing widows and orphans with lua-widow-control”, *TUGboat* 43:1

Max Chernoff

A request from *Zpravodaj*, the journal of the Czech/Slovak TEX group, to republish the subject article led to these updates. The section numbers here correspond to those in the original article.

3.3 Clubs

In the original article, I discussed the origin of the typographical terms “widow”, “orphan”, and “club”. The first two terms are fairly well-known, but I had this to say regarding the third:

> *The TEXbook* never refers to “orphans” as such; rather, it refers to them as “clubs”. This term is remarkably rare: I could only find a *single* source published before *The TEXbook* — a compilation article about the definition of “widow” — that mentions a “club line” [...]
>
> I spent a few hours searching through Google Books and my university library catalogue, but I could not find a single additional source. If anyone has any more information on the definition of a “club line” or why Knuth chose to use this archaic Scottish term in TEX, please let me know!

Conveniently, Don Knuth — the creator of TEX — read my plea and sent me this reply:

> I cannot remember where I found the term “club line”. Evidently the books that I scoured in 1977 and 1978 had taught me only that an isolated line, caused by breaking between pages in the midst of a paragraph, was called a “widow”; hence TEX78 had only “`\chpar4`” to change the “`widowpenalty`”. Sometime between then and TEX82 I must have come across what appeared to be an authoritative source that distinguished between widows at the beginning of a paragraph and orphans or club lines at the end. I may have felt that the term “orphan” was somewhat pejorative, who knows?

So this (somewhat) resolves the question of where the term “club” came from.

9 Options

The overview to the “options” section stated that:

> Plain TEX/OpTEX Some options are set by modifying a register, while others must be set manually using `\directlua`.

However, this is no longer true. Now, commands are provided for all options in all formats, so you no longer need to use ugly `\directlua` commands in your documents. The old commands still work, although they will likely be removed at some point in the future.

9.5 Penalties

`\brokenpenalty` now also exists as a LATEX and ConTEXt key. lua-widow-control will pick up on the values of `\widowpenalty`, `\clubpenalty`, and `\brokenpenalty` regardless of how you set them, so the use of these dedicated keys is entirely optional.

9.6 `\nobreak` behaviour

The Plain/OpTEX command is now:

`\lwcnobreak{`⟨*value*⟩`}`

9.8 Draft mode

Since v2.2.0, lua-widow-control has a “draft mode” which shows how lua-widow-control processes pages.

Plain TEX/OpTEX	`\lwcdraft 1`
LATEX	`\lwcsetup{draft}`
ConTEXt	`\setuplwc[draft=start]`

Draft mode has been used for typesetting this article. It has two main features:

First, it colours lines in the document according to their status. Any remaining widows and orphans will be coloured red, any expanded paragraphs will be coloured green, and any lines moved to the next page will be coloured blue.

Second, this draft mode shows the paragraph costs at the end each paragraph, in the margin.

This draft mode leads to a neat trick: if you don’t quite trust lua-widow-control, or you’re writing a document whose final version will need to be compilable by both pdfLATEX and LuaLATEX, you can load the package with:

```
\usepackage[draft, disable]
           {lua-widow-control}
```

This way, all the widows and orphans will be coloured red and listed in your log file. When you go through the document to try and manually remove the widows and orphans — whether through the `\looseness` trick or by rewriting certain lines — you can easily find the best paragraphs to modify by looking at the paragraph costs in the margins. If you’re less cautious, you can compile your document with lua-widow-control enabled as normal and inspect all the green paragraphs to see if they look

doi.org/10.47397/tb/43-3/tb135chernoff-lwc

acceptable to you.

You can also toggle the paragraph colouring and cost displays individually:

Plain TEX/ OpTEX	`\lwcshowcosts 1` `\lwcshowcolours 0`
LATEX	`\lwcsetup{showcosts=true}` `\lwcsetup{showcolours=false}`
ConTEXt	`\setuplwc[showcosts=start]` `\setuplwc[showcolours=stop]`

To demonstrate the new draft mode, I have tricked lua-widow-control into thinking that every column in this article ends in a widow, even when they actually don't. This means that lua-widow-control is attempting to expand paragraphs on every column. This gives terrible page breaks and often creates new widows and orphans, but it's a good demonstration of how lua-widow-control works.

10 Presets

The original article stated that "presets are LATEX-only". However, lua-widow-control now supports presets with both LATEX and ConTEXt using the following commands:

LATEX	`\lwcsetup{`⟨*preset*⟩`}`
ConTEXt	`\setuplwc[`⟨*preset*⟩`]`

11 Compatibility

This quote:

> It doesn't modify [...], inserts/floats,

isn't strictly true since v2.1.2 since lua-widow-control now handles moving footnotes.

This statement is also no longer true:

> there are a few issues with ConTEXt [...] lua-widow-control is inevitably more reliable with Plain TEX and LATEX than with ConTEXt.

All issues with ConTEXt — including grid snapping — have now been resolved. lua-widow-control should be equally reliable with all formats.

11.1 Formats

In addition to the previously-mentioned formats/engines, lua-widow-control now has preliminary support for LuaMetaLATEX and LuaMetaPlain.[1] Aside from a few minor bugs, the LuaMetaLATEX and LuaMetaPlain versions work identically to their respective LuaLATEX versions. With this addition, lua-widow-control now supports seven different format/engine combinations.

[1] github.com/zauguin/luametalatex

11.3 Performance

Earlier versions of lua-widow-control had some memory leaks. These weren't noticeable for small documents, although it could cause slowdowns for documents larger than a few hundred pages. However, I have implemented a new testing suite to ensure that there are no memory leaks, so lua-widow-control can now easily compile documents > 10 000 pages long.

13.4 Footnotes

Earlier versions of lua-widow-control completely ignored inserts. This meant that if a moved line had associated footnotes, lua-widow-control would move the "footnote mark" but not the associated "footnote text". lua-widow-control now handles footnotes correctly through the mechanism detailed in the next section.

13.4.1 Inserts

Before we go into the details of how lua-widow-control handles footnotes, we need to look at what footnotes actually are to TEX. Every `\footnote` command ultimately expands to something like `\insert`⟨*class*⟩`{`⟨*content*⟩`}`, where ⟨*class*⟩ is an insertion class number, defined as `\footins` in this case (in Plain TEX and LATEX). Inserts can be found in horizontal mode (footnotes) or in vertical mode (`\topins` in Plain TEX and floats in LATEX), but they cannot be inside boxes. Each of these insert types is assigned a different class number, but the mechanism is otherwise identical. lua-widow-control treats all inserts identically, although it safely ignores vertical mode inserts since they are only ever found between paragraphs.

But what does `\insert` do exactly? When TEX sees an `\insert` primitive in horizontal mode (when typesetting a paragraph), it does two things: first, it processes the insert's content and saves it invisibly just below the current line. Second, it effectively adds the insert content's height to the height of the material on the current page. Also, for the first insert on a page, the glue in `\skip`⟨*class*⟩ is added to the current height. All this is done to ensure that there is sufficient room for the insert on the page whenever the line is output onto the page.

If there is absolutely no way to make the insert fit on the page — say, if you placed an entire paragraph in a footnote on the last line of a page — then TEX will begrudgingly "split" the insert, placing the first part on the current page and "holding over" the second part until the next page.

There are some other TEXnicalities involving

`\count`⟨*class*⟩ and `\dimen`⟨*class*⟩, but they mostly don't affect lua-widow-control. See Chapter 15 in *The TEXbook* or some other reference for all the details.

After TEX has chosen the breakpoints for a paragraph, it adds the chosen lines one by one to the current page. Whenever the accumulated page height is "close enough" to the target page height (normally `\vsize`) the `\output` token list (often called the "output routine") is expanded.

But before `\output` is called, TEX goes through the page contents and moves the contents of any saved inserts into `\vbox`es corresponding to the inserts' classes, namely `\box`⟨*class*⟩, so `\output` can work with them.

And that's pretty much it on the engine side. Actually placing the inserts on the page is reserved for the output routine, which is defined by the format. This too is a complicated process, although thankfully not one that lua-widow-control needs to worry about.

13.4.2 LuaMetaTEX

The LuaMetaTEX engine treats inserts slightly differently than traditional TEX engines. The first major difference is that insertions have dedicated registers; so instead of `\box`⟨*class*⟩, LuaMetaTEX has `\insertbox`⟨*class*⟩; instead of `\count`⟨*class*⟩, LuaMetaTEX has `\insertmultiplier`⟨*class*⟩; etc. The second major difference is that LuaMetaTEX will pick up inserts that are inside of boxes, meaning that placing footnotes in things like tables or frames should mostly just work as expected.

There are also a few new parameters and other minor changes, but the overall mechanism is still quite similar to traditional TEX.

13.4.3 Paragraph breaking

As stated in the original article, lua-widow-control intercepts TEX's output immediately before the output routine. However, this is *after* all the inserts on the page have been processed and boxed. This is a bit of a problem because if we move a line to the next page, we need to move the associated insert; however, the insert is already gone.

To solve this problem, immediately after TEX has naturally broken a paragraph, lua-widow-control copies and stores all its inserts. Then, lua-widow-control tags the first element of each line (usually a glyph) with a LuaTEX attribute that contains the indices for the first and last associated insert. lua-widow-control also tags each line inside the insert's content with its corresponding index so that it can be found later.

13.4.4 Page breaking

Here, we follow the same algorithm as in the original article. However, when we move the last line of the page to the next page, we first need to inspect the line to see if any of its contents have been marked with an insert index. If so, we need to move the corresponding insert to the next page. To do so, we unpack the attributes value to get all the inserts associated with this line.

Using the stored insert indices and class, we can iterate through `\box`⟨*class*⟩ and delete any lines that match one of the current line's indices. We also need to iterate through the internal TEX box `hold_head` — the box that holds any inserts split onto the next page — and delete any matching lines. We can safely delete any of these lines since they are still stored in the original `\insert` nodes that we copied earlier.

Now, we can retrieve all of our previously-stored inserts and add them to the next page, immediately after the moved line. Then, when TEX builds that page, it will find these inserts and move their contents to the appropriate boxes.

16 Known issues

The following two bugs have now been fully resolved:

- When running under LuaMetaTEX, the log may contain [...]
- TEX may warn about overfull `\vbox`es [...]

The fundamental limitations previously listed still exist; however, these two bugs along with a few dozen others have all been fixed since the original article was published. At this point, all *known* bugs have been resolved; some bugs certainly still remain, but I'd feel quite confident using lua-widow-control in your everyday documents.

There is, however, one new issue:

- lua-widow-control won't properly move footnotes if there are multiple different "classes" of inserts on the same line. To the best of my knowledge, this shouldn't happen in any real-world documents. If this happens to be an issue for you, please let me know; this problem is relatively easy to fix, although it will add considerable complexity for what I think isn't a real issue.

◇ Max Chernoff
mseven (at) telus dot net
https://ctan.org/pkg/lua-widow-control