Processing paragraphs:

what happens there

and what do we want

TUG, June 28, 2010

- In TEX a paragraph (or any text) is a linked list of so called nodes. Think of glyph nodes, glue nodes, kern nodes.
- Nodes can originate from user actions or inserted by the engine itself. Math for instance is processed into nodes using an intermediate representation.
- Boxes are packed: horizontally or vertically (a paragraph).
- For each paragraph a long list is broken into smaller ones representing lines. Lines are hboxes and in between come others items like glue and penalties.
- As packaged boxes can be unboxed and injected in for instance the regular flow of a paragraph, an already processed list can be seen multiple times and get reprocessed.

Good to know

- It all happens within some size related constraints, like hsize, left- and rightskip, (hanging) indentation.
- There are also esthetic constraints, like preferred points for breaking or the opposite, or the overall quality of the result.
- Words can be broken into two parts (hyphenation) and special things might happen there.
- Breaking lines involves spacing, stretch, shrink, tolerance and more aspects.
- A paragraph can start out in one direction but directions can change (back) anywhere, this feature is inherited from Aleph/Omega.
- The last line can get a special treatment, something inherited from ε-TEX.
- Characters can protrude in the left or right margin and be widened or narrowed, introduced in pdfTEX.
- Ligatures can demand some special treatment when hyphenated, although one may wonder if we should bother about it as in practice it does not matter that much.

- In LuaTEX we can intercept the building and packaging of lists at multiple stages.
- This way one can change the order of normal actions or do additional things.
- For instance OpenType font features are dealt with in Lua by manipulating the node lists.
- In ConTEXt we have plugin mechanisms so that we can overload or extend mechanisms like the parbuilder of hpacker.
- So, we have the core TEX machinery, including all its normal functionality, but we can change it's behaviour.

How can we influence this

- The parbuilder is mostly one big function that has got quite complex in successive extended TEX's.
- By looking at the whole we can simplify statements and avoid goto's.
- We can replace web and C macros by constructs more natural to Lua.
- We can consider splitting the currently mixed phases.
- It makes sense to add more tracing.
- In Lua we can use different data structures and don't need to be clever with memory and access.

- Eventually in ConTEXt we will use these hooks to create more flexibile (and maybe better) solutions (replace existing code).
- Think for instance of more complex paragraph shapes or more complex flow of text around graphics (crossing pages).
- An example is the Oriental TEX related optimizer: here we manipulates lists and feed them into the regular parbuilder.
- Originally we thought that we needed to extend the parbuilder but in practice we ended up with an interplay between font technology and regular parbuilding.
- In the optimizer we reprocess the node lists quite drastically: we replace words by alternative renditions and then let the regular parbuilder do its work again.
- So, not all paragraph optimizations demand tweaking of the parbuilder but it's hard to predict what we will do in the future.