A Glimpse on Advanced LaTeX

J.M. Pérez Pardo

Contents

1 How LaTeX prints glyphs on a page

Before we answer this question we are going to look at how did a printing press (J. Gutenberg 1400a.d.) work. The glyphs were molded in small metals blocks like those appearing in Figure 1. These blocks were used to compose a page by inserting them in a wooden chase, see Figure 1. First building words and then lines. The full chase was then covered with ink and introduced in the *press* to print a page.

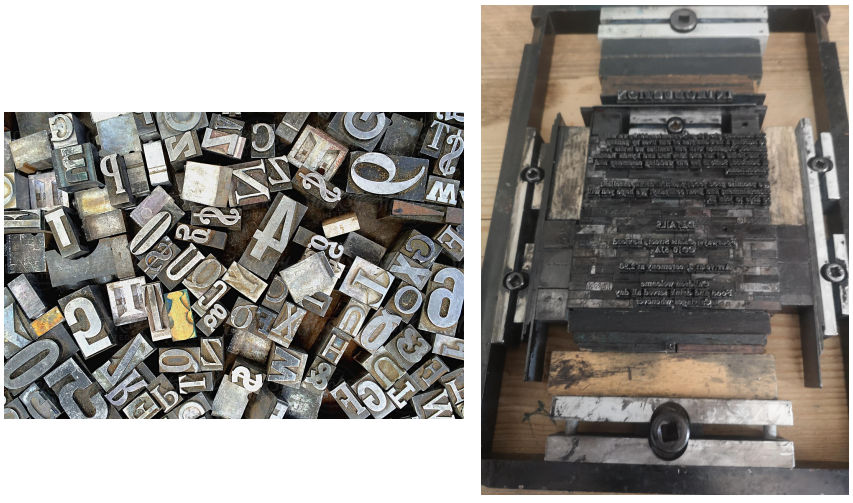


Figure 1: *Left*: Metal blocks and chase used in a printer press. *Right*: Chase where the metal blocks are inserted

TEX (and LATEX.) does actually a very similar thing to compose a document. Each glyph is contained in an invisible container called *box* that has a rectangular shape. This box has given dimensions, *width*, *height* and *depth* and determines how much space does it take. The *ink* of the glyph is adjusted with respect to the box. The ink might even lie outside of the box. Actually TEX does not care about the ink. All that matters to it are the boxes and their alignment. How the ink is placed with respect to the box is decided by the persons how designed the different fonts. When parsing the document, TEX construct words putting all these boxes together and building bigger boxes.

2 Boxes

A box is a rectangular shaped object that has three numbers (also called dimensions) associated to it, *width*, *height* and *depth*. These dimensions determine how big a box is and what its relative position to neighboring boxes will be. The structure of a box can be seen in Figure 2. The baseline determines the imaginary line where characters lie. The height determines how high a character lies above the baseline and the depth determines how much it *hangs* from the baseline. For instance, the box of the glyph "h" has a bigger height than the box of the glyph "a" and both of them have 0 depth. The box of the glyph "g" has the same height than the box of the glyph "a" but non-zero depth.

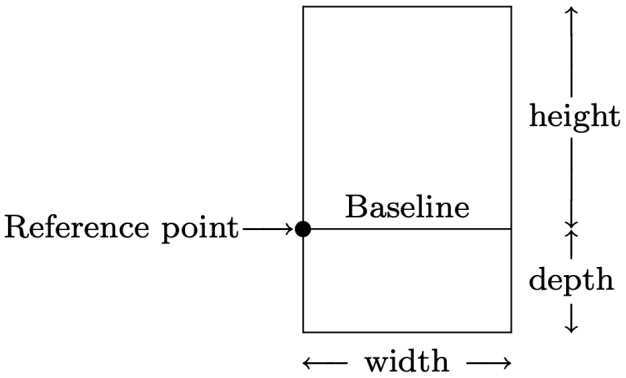


Figure 2: A box and its dimensions. *Source*: The TEXbook [1]

The macro \bx defined in the preamble displays the margins of the boxes so that we can have a look at the dimensions directly:

Actually these are too small to be appreciated in depth so I will use a larger font. This is done with the \Huge command.

Notice that "x", "M", "f" have no depth while "g", ")", "p", "f" have. In addition, notice that the ink of the italic shape of the letter "f" lies outside the box. This is something usual in italic fonts. Otherwise the characters in a word would be too separated from each other. Notice also how each character has a different width.

3 Horizontal and vertical modes. How LaTeX puts boxes together and prints pages.

Now that we know how LaTeX understands each character we are going to get an idea of how it does to build the page. When producing the document the source file is split into text units which are paragraphs. Each paragraph will end up being a paragraph in the final document. When the parser begins reading a paragraph it knows that the paragraph ends when a \par command is found. This might be the first time that we see this command because it is rarely typed in the source code. The reason is that a blank line in the source code, two end of line characters with possibly white spaces between them, is interpreted automatically like the \par command. The next two paragraphs are an example of this. Check the source code. There is no blank line in between the next two paragraphs but a \par command.

Two paragraphs separated by a \par

No one expects the Spanish Inquisition! Our chief weapon is surprise, fear and surprise; two chief weapons, fear, surprise, and ruthless efficiency! [a \par command is here and no blank line]

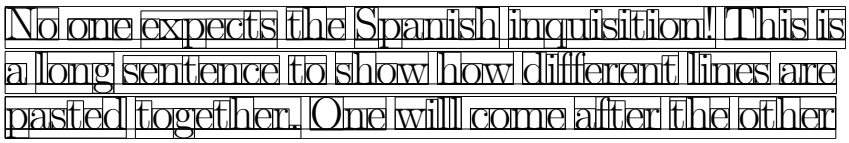
Er, among our chief weapons are: fear, surprise, ruthless efficiency, and near fanatical devotion to the Pope! Um, I'll come in again

Whenever a paragraph starts or ends several things happen. There are some obvious formatting operations that occur, like adding the indentation on the first line of including extra space between the first line of a paragraph and the last line of the previous one. But there are many other things that occur internally and we will give a brief description of what happens with them. Before a paragraph starts the parser is in the so called *vertical mode*. When the parser meets the first character of a paragraph it switches to horizontal mode and starts putting boxes together. First it collects all the non-white space characters that he finds up to the next white space character. The boxes are aligned at their reference points, so that they share the baseline. Once the word is complete a new box is defined that has height and depth the biggest height and depth of the boxes that compose it, and width the total sum of the widths of all the boxes. This is illustrated next with the word "Perspective".

The different boxes of the words are then put together. Between the words, white space is added. As you can see in the example below white space is not a box of a given width without ink. This white space is called *glue* and is what

allows TEX to print the justified blocks of text. We will speak about *glue* later.

TEX will continue to do so until it has finished parsing the full paragraph. At this time what TEX has is a very long line representing the paragraph. It is now time to break the paragraph into lines. The *glue*, i.e., the white space between words, has not a fixed length but instead can stretch or shrink to make the lines exactly as long as needed. TEX has a complicated algorithm, cf. [1, Chapter 14], to decide where are the optimal points to break the paragraph into lines such that the outcome is as visually pleasant as possible. We are not going to enter in the details of this algorithm but at the end of this section we will discuss some basic things that will allow us to tune how it works. After the paragraph is broken into lines the parser goes into *vertical mode*. This time, the boxes that make the lines are put vertically one after the other and there is also some *glue* inserted between these boxes. You can get an idea with the following picture.



All that is missing is to stretch the glue to make all line boxes have the same length. The parser continues then in *vertical mode* until the next paragraph starts. When there are enough lines to fit a page it is *shipped* for printing. In principle TEX could use a similar algorithm than it uses for breaking a paragraph into lines to split the document into pages but it doesn't. When the lines have enough length to fit the page they are stretched or shrunk for the best output and the page is closed for shipping. This was done so because of memory considerations. Notice that the full document should be in memory to accomplish this and for a very large document this could be a problem.

We are going to give now some hints in how the paragraph breaking algorithm works. The fundamental idea is that breaking the paragraph at a point has a cost. This was called by D.E. Knuth the *badness*. The algorithm considers many possibilities of breaking the paragraph into lines and chooses the one with the minimum *badness*. This works roughly as follows. If a line has to be stretched or shrunk to fit the page it gets a *penalty* depending on how much it had to be stretched or shrunk. Other operations like hyphenating a word also have a *penalty* and so on. The interesting thing about this is that penalties can be set by hand to force or prevent TEX to take a line break. This is done with the \penalty command. Normally you do not need to take care about this but on occasions you might feel that a certain break is not appropriate and we will see what can be done in those cases. Typical penalties handled by TEX are in a range between 10 and 10. A penalty of 10000 (\penalty 10000) or more is considered to be so large that TEX will never break there. Penalties can be negative too. This makes certain breakpoints to be more desirable than others as the total badness will be diminished if the break point happens there. A penalty of -10000 (\penalty -10000) is considered to be so low that TEX will always break there.

A paragraph to be used as reference to see the effect of adding penalties.

No one expects the Spanish Inquisition! Our chief weapon is surprise, fear and surprise; two chief weapons, fear, surprise, and ruthless efficiency! Er, among our chief weapons are: fear, surprise, ruthless efficiency, and near fanatical devotion to the Pope! Um, I'll come in again...

As you can see the points that have been chosen by TEX to break the lines are after the words "and", "our", and "to". The next paragraph has a penalty of 10000 after the word "our".

The paragraph above with a penalty of 10000 after the word "our"

No one expects the Spanish Inquisition! Our chief weapon is surprise, fear and surprise; two chief weapons, fear, surprise, and ruthless efficiency! Er, among our chief weapons are: fear, surprise, ruthless efficiency, and near fanatical devotion to the Pope! Um, I'll come in again...

You can see that the break didn't occur after the word "our" but there were more changes in the paragraph. To avoid stretching that line too much TEX changed the breaks of the other lines too. We want to keep the "and" as the breaking point of the first paragraph so let us add a penalty of -10000 after the "and" and see what happens.

The paragraph above with a penalty of 10000 after the word "our" and a penalty of -10000 after the word "and"

No one expects the Spanish Inquisition! Our chief weapon is surprise, fear and surprise; two chief weapons, fear, surprise, and ruthless efficiency! Er, among our chief weapons are: fear, surprise, ruthless efficiency, and near fanatical devotion to the Pope! Um, I'll come in again...

This time the white space of the second line was stretched a lot to fit the line width. Notice that not all white spaces were stretched by the same amount... We will understand why this is so when we learn about glue in the next section. Let us consider a final example with a penalty -10000 after the word "fear,", that is, about at the middle of one optimal line.

The paragraph above with a penalty of -10000 after the word "fear,".

No one expects the Spanish Inquisition! Our chief weapon is surprise, fear and surprise; two chief weapons, fear, surprise, and ruthless efficiency! Er, among our chief weapons are: fear, surprise, ruthless efficiency, and near fanatical devotion to the Pope! Um, I'll come in again...

As expected we are forcing a line break there, but the line is stretched by a fairly large amount. Actually, this is almost what TEX to force a line break when one types \\ , but the stretching needs to be corrected.

4 The *glue* and spacing in LaTeX

All the spacing commands in LaTeX are related with *glue*. When parsing the document any amount of spaces between characters is substituted by *glue* and a single end of line character is transformed by the parser into a white space and therefore also *glue*. Actually the name *glue* is not very descriptive of what it does and the name *spring* would be more appropriate. This is how D.E. Knuth puts it: "*Once you understand TeX's concept of glue, you may well decide that it was misnamed; real glue doesn't stretch or shrink in such ways, nor does it contribute much space between boxes that it welds together. Another word like "spring" would be much closer to the essential idea, since springs have a natural width, and since different springs compress and expand at different rates under tension. But whenever the author has suggested changing TeX's terminology, numerous people have said that they like the word "glue" in spite of its inappropriateness; so the original name has stuck.*"

There are two types of *glue*. One for the horizontal mode and one for the vertical mode. The primitive TeX commands for including them are \hskip and \vskip. They add glue respectively in the horizontal and the vertical mode. Glue accepts three parameters, which must be *dimensions*. That is a number following a unit of measure like `pt` (points), `cm` (centimeters), `ex` (the width of a small x in the current font), `em` (the width of a capital M in the current font) and others.

```
\hskip <dimen> plus <dimen> minus <dimen>
\vskip <dimen> plus <dimen> minus <dimen>
```

The first dimension gives the fix space that the glue will take. The second dimension, after the "plus", determines how much it can stretch and the third one, after the "minus", how much it can shrink. For example, in the source code you could write

```
WORD \hskip 2em plus 1ex minus 2pt WORD
```

Which results in

WORD WORD

This means that each *glue* between words has a determined amount of stretchability or shrinkability. When a box has to be stretched (or shrunk) each glue in it adds (or substracts) to each fixed length a quantity proportional to its glue plus (minus) dimension to get the desired length. For instance, after punctuation marks the *glue* is more stretchable than between words and so is *glue* after a dot than after a comma. This explains the behaviour of the stretching in the examples of the previous section.

In addition to the dimensions mentioned above, TeX has defined **three different types of infinity**. Each one of them is infinitely larger than the previous one. These are called `fil`, `fill` and `filll` in increasing order of infinity. These infinities can be used as the dimension of the stretching or shrinking quantity in *glue*. In essence they provide an infinite amount of stretchability that fills the line, justifying their name. Below some examples on how this works.

Using *glue* with infinite stretchability

Below we have several examples. There is a line with the source code and directly below it is the output that it produces.

```
word \hskip 0pt plus 1fil minus 0pt word:
word                              word

word \hskip 0pt plus 1fill minus 0pt word:
word                                                              word

word \hskip 0pt plus 1filll minus 0pt word:
word                                                              word

word \hskip 0pt plus 1fill  word  \hskip 0pt plus 1fill word:
word                            word                              word

word \hskip 0pt plus 1fill  word  \hskip 0pt plus 2fill word:
word                    word                                      word

word \hskip 0pt plus 1filll  word  \hskip 0pt plus 10fill word:
word                                                        word word
```

We can see that a greater infinity type neglects the presence of lower infinities. Infinities of the same type will be weighted according to the amount of it in the line. There is just one point that should raise our attention tight now. Why doesn't the infinite stretchability of the first example push the right word to the end of the line like in the other cases? The answer has to do with what TEX does at the end of a paragraph to prevent the last line to stretch. At the end of a paragraph automatically an "\hskip `0pt plus 1fil minus 0pt`" is inserted to keep the line short. The second word is pushed at its two sides by equally weighted infinite *glue*. That is why it remains near the middle of the line.

There are some macros which are shorthand for this infinite stretchable *glue*. For instance \hfil is defined as \hskip `0pt plus 1fil minus 0pt` and equivalently \hfill, \vfil and \vfill. There is no macro for the greatest infinity because it is rarely used. There are also macros to define fixed, i.e., non stretchable/shrinkable *glue*. These are \hspace{<*dimen*>} and \vspace{<*dimen*>}. The former is defined by \hskip `<dimen> plus 0pt minus 0pt` and the latter is defined equivalently. In this document I have used these macros and also some *glue* to tune the adjustment of the figures. Have a look at them in the source file to see what effect they had in the final layout.

There is still one important thing to discuss and is that vertical *glue* takes effect only during the vertical mode. When TEX is in horizontal mode and encounters a \vspace command it is not inserted immediately but after the line that contains it and will be used in vertical mode to spread the lines. Check the following example where a \vspace command is used in the middle of the paragraph. Intuitively one would expect that after the command the rest of the text would have been shifted, but this is not what happens.

A \vspace{1cm} is inserted after "Inquisition!"

No one expects the Spanish Inquisition!Our chief weapon is surprise, fear and

surprise; two chief weapons, fear, surprise, and ruthless efficiency!

Er, among our chief weapons are: fear, surprise, ruthless efficiency, and near fanatical devotion to the Pope! Um, I'll come in again...

To prevent this behaviour use vertical *glue* at the beginning or end of a paragraph, or between paragraphs.

5 Special commands to manipulate boxes

It is time know to learn some commands that allow to manipulate boxes. Knowing this allows for fine tuning of placement, in particular in formulas. LaTeX does a great job at typesetting but not all the cases are covered by the programers and many cases need to be adjusted by hand. Many of these situations can be addressed easily by knowing the following commands. In addition, they can be used to defined many useful macros that can be adapted to many situations. We will learn this in the examples. Remember that arguments in square brackets represent optional arguments, they might be given or not. Each command comes with some examples. The boxes are typeset using the \bx command defined in the preamble. They would not be seen otherwise.

\hbox{contents} This is a TeX primitive. It creates a box with the contents parsed in *horizontal mode*. It should be avoided in favour of \mbox except in special situations. This box is unbreakable once created.
Example: `\hbox{Hello}`

Hello

\vbox{contents} This is a TeX primitive. It creates a box with the contents parsed in *vertical mode*. Should be used in situations when one wants to put one box on top of other box. The minipage environment should be used instead of this one.
Example: `\vbox{a\\b\\c} \vbox{\hbox{a}\hbox{b}\hbox{c}}`

a
b
c

a
b
c

\mbox{contents} Essentially like \hbox with more predictable behaviour.
Example: `\mbox{Hello}`

Hello

\makebox[width][alignment]{contents} Like \mbox but defines a box with given width. The contents can be aligned l,c or r.
Example: `\makebox[1em][r]{Hello} \makebox[5em][r]{Hello}`

Hello Hello

```
\makebox[5em][c]{Hello} \makebox[5em][l]{Hello} \makebox[1em][l]{Hello}
```

Hello Hello Hello

\newsavebox{\name} Creates a *register*, that is, a bin, where one can save a box.

\sbox{\name}{contents} Creates a box in horizontal mode and saves it in a register (bin) with name \name that was previously created with \newsavebox.
Example:

```
\newsavebox{\mybox}
\sbox{\mybox}{Hello}
The width of \bs{mybox} is \mbox{\the\wd\mybox}.
The height of \bs{mybox} is \mbox{\the\ht\mybox}.
The depth of \bs{mybox} is \mbox{\the\dp\mybox}.
```

The width of \mybox is 22.4945pt. The height of \mybox is 6.8872pt. The depth of \mybox is 0.0pt.

\savebox[width][alignment]{\name}{contents} Like \sbox but adjust the width an alignment like \makebox.

\usebox{\name} Prints the box in the register \name.
Example: `\usebox{\mybox}`

Hello

\raisebox{dimension}[height][depth]{contents} raises the contents by dimension. This is, the baseline of the box is moved downwards (or upwards if dimension is negative). The resulting box can have the given height and depth.
Example:

```
\bx{\raisebox{3pt}{Hello}}Hello
\bx{\raisebox{-3pt}{Hello}}Hello
\bx{\raisebox{3pt}[5pt][2pt]{Hello}}Hello
```

Hello Hello Hello Hello Hello Hello

\rule[raise]{width}{thickness} Creates a box fully covered with ink. If width or thickness is zero it will not be visible in the document, but the box will occupy some space.
Example: `\rule{1em}{1ex} \rule{10em}{0pt} \rule[5pt]{1em}{1ex}`

Notice that the rule in the middle can be seen because of the \bx command. Otherwise it would be invisible.

\llap{contents} Collapses the box of contents to a vertical line at the right. In math mode use \mathllap instead, which is provided by the `mathtools` package.

\clap{contents} Collapses the box of contents to a vertical line at the center. In math mode use \mathclap instead. Both commands are provided by the `mathtools` package.

\rlap{contents} Collapses the box of contents to a vertical line at the left. In math mode use \mathrlap instead, which is provided by the `mathtools` package.
Example:

```
\hfil
\llap{\Huge a}
\hfil
\rlap{\Huge a}
\hfil
\clap{\Huge a}
\hfil
```

a a a

Notice the differences between these formulas. Can you guess what we did in the second one?

$$X = \sum_{1\leq i\leq j\leq n} X_{ij} \qquad\qquad X = \sum_{1\leq i\leq j\leq n} X_{ij}$$

\smash{contents} Collapses the box of contents to a horizontal line at the bottom. Works in text mode and math mode.
Example: `\smash{Hello}`

Hello

 Creates a box with the dimensions of contents but without ink. Works in text mode and math mode.
Example: `\phantom{Hello}`

References

[1] D. Knuth, *The TEXbook*, Addison-Wesley, 1990.