The wrapfig2 package

Claudio Beccari

Abstract Package wrapfig is extremely useful, but is delicate. It has many limitations because it has to compute the space in terms of number of lines necessary to indent the surrounding text so that it can wrap the insertion. Its documentation explains the limitations the users should be attentive to.

This short paper describes a new package, wrapfig2, that is backwards compatible with the original wrapfig, but aims to let the users have a better control on the number of lines. At the same time it defines a new environment, wraptext, that exploits the same ideas in order to insert a framed text block on a shaded background wrapped by the surrounding text.

Sommario Il pacchetto wrapfig è molto utile, ma è delicato. Presenta diverse limitazioni dovute alla difficoltà di calcolare correttamente di quante righe deve essere il rientro che consenta l'inserimento del materiale da contornare con il testo.

Questo breve articolo descrive un nuovo pacchetto, wrapfig2, retrocompatibile con il pacchetto originale wrapfig, che consente all'utente di avere un controllo migliore sul numero di righe del rientro del testo circostante. Contemporaneamente definisce un nuovo ambiente, wraptext per inserire un blocco di testo incorniciato e su uno sfondo sfumato, in un testo che lo circonda.

1. Introduction

Package wrapfig by Donald Arseneau has been available for several years; the last upgrade to our best knowledge dates back to 2003; it is very useful to insert small objects (figures, tables, and similar ones) in an indented part of the normal text; it is possible to configure the insertion so as to specify the overhang into the margin, the total width of the insert, even the total height of the necessary indention; it is possible to specify the margin where to protrude the insertion and even to float the inserted material. The available environments are wrapfigure, and wraptable; they depend on wrapfloat⟨*float kind*⟩[1], where the ⟨*float kind*⟩ is already available.

Such insertions may be fixed or floated; placement codes may be specified by the user.

The general syntax of this environment is the following (the example refers to a figure, but the syntax is similar for tables and other objects):

```
\begin{wrapfigure}[⟨total number of indented lines⟩]
    {⟨position⟩}[⟨overhang⟩]{⟨insert width⟩}
    ⟨material to be inserted⟩
\end{wrapfigure}
```

It should be sufficient; actually if the insert is pretty long the environment may have difficulties in computing the correct number of indented lines; the reasons are explained in the wrapfig package documentation; its author warns the user about the package idiosyncrasies.

1. Users are familiar with the figure and table float kinds, but by means of the float package it is possible to define other kinds.

Most of these idiosyncrasies can be avoided by choosing the right place in the source file where to specify the environment: it should not follow too close a page break, a sectioning command, a list, and so on. But according to our experience one of the most annoying features is the difficulty of counting the correct number of indented lines; we found that it is simpler to count the (generally small) number of lines to add to or subtract from the line number computed by the original software.

There are a couple of reasons for following this corrective approach, rather than the original approach.

1. The first time any wrapped insertion is specified, the users have no idea of how many indented lines are necessary; therefore they do not specify the optional argument ⟨*total number of indented lines*⟩; they let the software compute the necessary number of such lines. More often than not such number is not the most suitable one, and often the white space below the insertion is too high; or it may happen that the computed number of lines is too small so that the inserted material overlaps the neighbouring text. In the first case it is necessary to diminish the computed number of lines, while in the second case such number should be increased.
2. It is simpler to count the small number of extra indented lines, or the small number of overlapped full lines.

2. How to compute the correction line number

The new language LaTeX 3 offers many possibilities. Since the fall of 2020, most, if not all, the xparse functionalities are included into the LaTeX kernel, and, except in special (deprecated) cases, it is not necessary to load any package: the xparse documentation is still available, but the package is not explicitly required. This wrapfig2 uses one of the *deprecated* cases, therefore it provides to load such package.

On the opposite, the computing LaTeX 3 functionalities are not yet included into the kernel; therefore the xfp package is loaded by wrapfig2; among such functionalities the `\fpeval` function can operate also on length values and can perform several kinds of rounding operations.

We found more efficient to redefine known environments, or define new ones, with similar syntaxes; we therefore redefined wrapfigure and wraptable and defined the new wraptext environment.

The core of the computation of the corrective line number is the following.

1. By using the environment arguments, store the object to be wrapped into a box and determine its height.
2. Use the `\fpeval` to compute the ratio of the above height to the current base line height, and let `\fpeval` round such ratio to the nearest integer.
3. Add a small fixed correction number and eventually add the user specified positive or negative correction value.
4. Pass such computed number of lines as the first optional argument to wrapfloat with the specific ⟨*float kind*⟩ string.

Following the above policy, both original wrapping environments are redefined as described in the following subsections.

2.1. *Necessary packages and preliminary settings*

The redefinitions we devised require some packages, therefore the following ones are required and the new package provides to load them if they are not already loaded.

xfp provides the powerful LaTeX 3 commands `\fpeval` and `\inteval`; the former is used to make calculations and round the results to a specified number of fractional decimal digits. The second is similar but operates on integer operands.

curve2e is going to be used with the wraptext environment so as to draw the emphasising frame and a coloured background around the text. See the specific subsection below.

etoolbox is always handy when code has to be written so as to render it more easily readable and maintained.

3. Wrapped figures

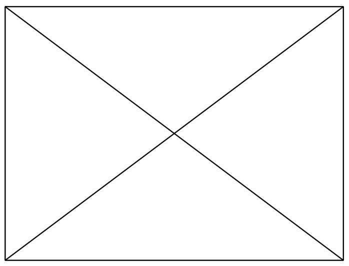
Figure 1. A rectangle with diagonals

The main redefinition of the wrapfigure, wrspatable and wraptext environments is commented in a further section that describes the relevant parts of the code; here we are more interested to show what is possible to do.

Let us first describe the syntax of these revised or new environments. The syntax shown in the box below refers to version 6 of this package. Some options specify how to select fallback settings in order to get the functionalities of versions 5 or 4). Please read the wrapfig2 package documentation.

```
\begin{wrapfigure}[⟨total indented lines number⟩]{⟨position⟩}[⟨overhang⟩]{⟨width⟩}⟨star⟩
⟨inserted figure⟩
\end{wrapfigure}

\begin{wraptable}[⟨total indented lines number⟩]{⟨position⟩}[⟨overhang⟩]{⟨width⟩}⟨star⟩
⟨inserted table⟩
\end{wraptable}

\begin{wraptext}<⟨indented lines number correction⟩>{⟨position⟩}[⟨overhang⟩]{⟨width⟩}
⟨optional style settings⟩
⟨inserted text block with other settings and corner radius of curvature⟩
\end{wraptext}
```

The package code for inserting a figure is the following; the one for inserting a table is very similar.

```
\NewDocumentEnvironment{wrapfigure}{o m o G{\z@}}%
    {\wrapfloat{figure}[#1]{#2}[#3]{#4}}%
    {\endwrapfloat}
```

As it is can be seen from the code, the environment contents is first boxed into the box register `\NWFbox` with the help of the lrbox environment. Then, depending on the presence of the optional asterisk the total number of lines of the object to be included is measured; if the asterisk is present the corrective argument is algebraically added to the estimated height; if the asterisk is absent then if the total number of lines is specified, such a number is used as in the original environment; if the user did not specify any value, the number of indented lines is evaluated by the original environment. A specific figure might be inserted by one of the following statements

```
\begin{wrapfigure}{l}{50mm} figure code \end{wrapfigure}
\begin{wrapfigure}[10]{l}{50mm} figure code \end{wrapfigure}
\begin{wrapfigure}[3]{l}{50mm}* figure code \end{wrapfigure}
```

The first statement is possibly the first one the users write in their document; the second is the one that exploits the original environment functionalities, while the third uses just the correction value to add after controlling what the first statement failed to compute correctly.

4. Wrapped tables

first	second
third	fourth

With (small) tables the situation is very similar; the difference relies on the fact that small tables generally are not typeset with a prescribed width; generally the user is not so good in estimating a suitable width for a table that relies on the widths of all the tabular cells. Therefore the last mandatory argument of the wraptable opening statement may be *omitted* thanks to a feature of the xparse package that may use brace delimited optional arguments; if this argument is omitted, together with the braces, the software is capable of determining the object width and may provide accordingly.

The code for the redefinition of the new wraptable environment is not so different from that of wrapfigure.

```
\NewDocumentEnvironment{wraptable}{o m o G{\z@}}%
    {\wrapfloat{table}[#1]{#2}[#3]{#4}}%
    {\endwrapfloat}
```

It goes by itself that the user has to insert the whole table code using, for example, the tabular environment. Therefore the user has a wider range of possibilities to change the size of the table, as well to correct the estimated height of the actual table. Since the underlying code is the one of the wrapfloat environment, it is possible that the various packages that deal with tables and/or with their captions still work properly with the new definition. In any case the result is still acceptable as with figures.

Caution

In both redefinitions of the standard wrapping environments, if the optional asterisk is specified while the first optional argument is omitted, no error is raised and no warning is output; simply the presence of the asterisk is ignored and the wrapped figure or table is typeset with the original environments; if the authors use the asterisk, they should pay attention to enter into the first optional argument the *corrective* number of lines, not the absolute one.

5. Wrapped text

Recently a question was posted on `stackexchange` in order to get help for creating some sort of small framed medallions on a coloured background to be wrapped by text as well as a figure.

Text, text, text, text, text, text, text, text, text, text, text.

Of course it would not be too complex to create the medallion with the functionalities of the tcolorbox and saving the graphic result into a box to be inserted as a wrapped figure by means of the wrapfigure environment. But in this last version 6 of the wrapfig2 package we preferred to draw the frame and is background by means of the picture environment extended with the curve2e package functionalities.

The participants to that thread on `stackexchange` suggested the WrapText environment based on the use of the original wrapfigure. Unfortunately the vertical space computed by wrapfigure more often than not was too small, and the medallion bottom would overlap the following normal text. A good friend of ours asked us if it was possible to maintain the same structure based on wrapfigure, but with some means to correct the actual number of lines of the indented normal text.

```
\NewDocumentEnvironment{wraptext}{O{0} m O{0pt} G{0.5\columnwidth}}%
{%                                                         Open environment
  \insertwidth=#4\WFscalewidth
  \def\textplacement{#2}%
  \def\textcorrection{#1}%
  \def\textoverhang{#3}%
  \bgroup\edef\x{\egroup\noexpand\wrapfloat{text}%
    [\textcorrection]{\textplacement}[\textoverhang]{\insertwidth}*}\x%
  \def\caption{\unskip%                       Redefine the \caption command
    \refstepcounter\@captype
    \let\@tempf\@caption
    \unless\ifcsname @float@c@\@captype\endcsname
      \expandafter\expandafter\let
        \expandafter\@tempf\csname @float@c@\@captype\endcsname
    \fi
    \@dblarg{\@caption\@captype}%    Use the internal LaTeX kernel commands
    }%
}{%                                                        Close environment
\endwraptext\ignorespaces}%
```

The user command syntax is the following:

```
\begin{wraptext}[⟨indented lines number correction⟩]{⟨location⟩}[⟨overhang⟩]{⟨width⟩}
    ⟨optional colour settings⟩
  \includeframedtext[⟨insertion measure⟩]{⟨text to frame⟩}[⟨settings⟩][⟨radius⟩]
\end{wraptext}
```

The solution of this problem was the motivation for designing the environments described in this article. As we mentioned above, the syntax of this environment is slightly different from the other two, but the philosophy is the same; as a matter of fact all environments use the same wrapfloat environment behind the scenes. If the wrapped framed text needs a caption (something we discourage, since we believe that the wrapping text is such as to describe why that text has been wrapped) it is necessary to use the `\caption` command with its arguments; with the `texstackexchange` solution and its default parameters, the caption would turn out to be in the form "Figure 3.2", but the label "Figure" does not seem to be the most appropriate; with version 6 the default value for the caption label is "Text", but by means of the babel or polyglossia language handlers it is possible to adapt it to any language.

The above code is not simpler than the ones used for the redefinitions of wrapfigure and wraptable, not only because there is no need to be backwards compatible, but also because some parameters, such as the text box width, are preset. The environment allows the user to specify the text box width; nevertheless a width shorter than the default might pose some justification problems, while a longer one would be too intrusive; depending on the text to be wrapped and on the page layout, we recommend to maintain the text box width in the range from 40% to 60% of the current column width; remember that while typesetting in one column mode the `\columnwidth` value equals that of `\textwidth`.

6. Performance

We don't know if this new extended version wrapfig2, performs better than the original version by Donald Arseneau. We partially modified his code, in particular the commands that define the total height of the wrapped object, or the total indention of the wrapping lines. We adopted the robust definitions of the LaTeX 3 language.

But certainly the idiosyncrasies that Arseneau described in his original documentation are still there; since the original patches to render the wrapping procedure compatible with other classes and/or packages have not been modified, it is possible they have the same pros and cons.

What we noticed is that the two revised environments and the new one are performing pretty well even without resorting to the correction mechanism that was the main reason for upgrading the package; probably this is due to the different way of boxing the material to be wrapped, or to the well conceived approach used by Arseneau.

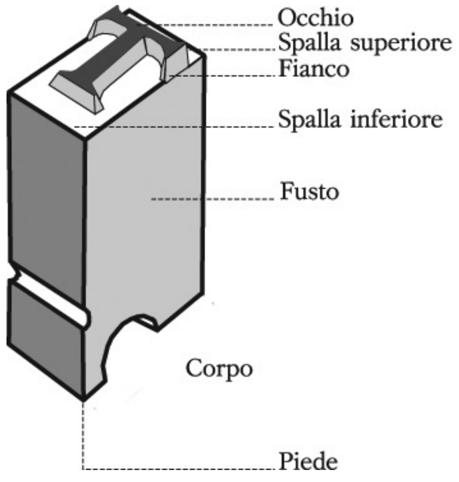

Figure 2. Italian names of a metal type details

The small examples we showed in the previous sections do not actually require any correction except for the wrapped figure; probably with taller objects the idiosyncrasies are more easily triggered. In any case the small wrapped figure 2 did not require any adjustment.

Notice, though, that the ⟨*width*⟩ of the object to be inserted refers to the true or estimated horizontal dimension of that object; if the width estimation is difficult, as with tabular material, it is better to specify a larger value and to specify `\centering` in order to insert the thinner object within the wider space that wrapfig2 reserves to the object. If the larger estimated width is too large, on a second compilation run it is possible to specify a more suitable value; when the specified width value is too small a black bar appears to the right of the inserted object. Even better: it is possible to omit the ⟨*width*⟩ specification; as we said before, the ⟨*width*⟩ parameter is a *braced optional argument* such that the software computes the necessary width quite well with figures and tables; this is what we did with figure 2.

In any case the inserted object must not have white margins around; for the wrapped text the built-in frame does not have any margins; for tables we already explained the difficulty of estimating their width, but they don't have any built-in white margins; the situation may be critical with images; it is important to carefully crop them so as to eliminate all four margins; resorting to the standalone package functionalities may be precious.

Acknowledgements

A hearty thank you goes to Donald Arseneau who first created the wrapfig package; many users are grateful to him for this package. I warmly acknowledge his original package paternity.

Heinrich Flech submitted me the described problems and suggested to implement more flexible correction solutions; I thank him very much.

I gratefully thank Lorenzo Pantieri who spotted some glitches in the first versions of this new package.

I also heartily acknowledge the suggestions concerning various testing sections, and some smart solutions to some problems received from Juan Luis Varona Malumbres.

Claudio Beccari
claudio.beccari@gmail.com