Making cutouts in paragraphs[*]

Peter Wilson[†]and Alan Hoenig   Maintained by David Carlisle[‡]

2021/10/13

Abstract

The cutwin package helps in making a cutout window in the middle of a paragraph.

Contents

1 Introduction

This manual is typeset according to the conventions of the LaTeX DOCSTRIP utility which enables the automatic extraction of the LaTeX macro source files [GM04].

Section 2 describes the usage of the cutwin package and commented source code is in Section 3.

2 The cutwin package

The code provided by the cutwin package is meant to help in creating windows, or cutouts, in a text-only paragraph. It is based on code originally published by Alan Hoenig [Hoe87].

---

[*] This file (`cutwin.dtx`) has version number v0.2, last revised 2021/10/13.

[†] Herries Press

[‡] https://github.com/latex-package-repositories/cutwin/issues

2.1 General

`\opencutleft`
`\opencutright`
`\opencutcenter`

Declarations specifying where a window is to be placed. The choices are: `\opencutleft` open into the left margin, `\opencutright` open into the right margin, and `\opencutcenter`, which is the default, open in the 'center' of the text, i.e, have text on both sides of the window.

`\cutfuzz`

This is provided as a convenience to reduce the number of overfull and underfull warnings. Its default definition is:

```
\newcommand{\cutfuzz}{%
  \vbadness=50000
  \hbadness=50000
  \sloppy}
```

and it is only applied to the paragraph being windowed.

2.2 Rectangular cutouts

A rectangular space can be placed in a paragraph with the text flowing across the gap. The space may break open into the top or side of the paragraph or, with some care, into the bottom (the number of lines specified for the cutout should not exceed the amount of text available for those lines). Some text or a logo or graphic may be placed within the window, or it may be left empty. In this instance I have put three short bold text lines in the window opening. The window should not be too wide as it can be difficult to track the exterior text lines across the gap.

Text
in
Window

`cutout`

The `cutout` environment, the body of which must be a single paragraph, enables a rectangular window to be cut out of the paragraph with the text flowing across the cutout. Use as:
`\begin{cutout}{`⟨*numtop*⟩`}{`⟨*leftwidth*⟩`}{`⟨*rightwidth*⟩`}{`⟨*numcut*⟩`}`
where ⟨*numtop*⟩ is the number of full lines above the window and ⟨*numcut*⟩ is the number of lines to be cut (giving the height of the window). The meaning of the lengths ⟨*leftwidth*⟩ and ⟨*rightwidth*⟩ depend on the location of the cutout:

- for a centered cutout ⟨*leftwidth*⟩ and ⟨*rightwidth*⟩ are the lengths of the text lines at the left and right sides of the window;
- for an open left cutout ⟨*leftwidth*⟩ is ignored and ⟨*rightwidth*⟩ is the length of the lines to the right of the cutout; and
- for an open right cutout ⟨*rightwidth*⟩ is ignored and ⟨*leftwidth*⟩ is the length of the lines at the left of the cutout.

`\pageinwindow`
`\windowpagestuff`

The macro `\pageinwindow` puts a zero-sized `picture` positioned at the left of the window aligned with the first line of the window (i.e, at the top left of the cutout). The `picture` consists of a `minipage` sized to fit the window. The contents of the minipage is `\windowpagestuff`. These two macros may be used to put a graphic or text into the windowed area.

The default definition of `\windowpagestuff` is:
`\newcommand*{\windowpagestuff}{}`
and you can change it as you wish. For instance, I used the following to put some text centrally within the above cutout.

```
\renewcommand*{\windowpagestuff}{%
  \centering\bfseries
  Text \\ in \\ Window \par}
```

You may well need to experiment to get everything adjusted to your satisfaction.

2.3 Shaped cutouts

A *shaped cutout* is one where the shape of the window is specified by the user who has to supply the length of the text lines bordering the cutout. Normally there is text on either side of the window but it could be open at either the left or right side of the paragraph. It is possible to put a logo or some text in the window. In this paragraph with a shaped cutout I have used a large $ sign as a simple logo.

$

`shapedcutout`

The `shapedcutout` environment, the body of which must be a single paragraph, enables an arbitrary shaped window to be cut out of the paragraph with the text flowing across the cutout. Use as:
`\begin{shapedcutout}{`⟨*numtop*⟩`}{`⟨*numcut*⟩`}{`⟨*shapespec*⟩`}` where ⟨*numtop*⟩ is the number of full lines above the window, ⟨*numcut*⟩ is the number of lines to be cut (giving the height of the window) and ⟨*shapespec*⟩ is the specification of the length of the lines bordering the cutout.

More precisely ⟨*shapespec*⟩ is a comma-separated list of the lengths of the text lines bordering the window.

- For a centered cutout one pair of entries are required for each cut line denoting the length of the left and right part of the cut line. There must be exactly ⟨*numcut*⟩ pairs.
- For example you might do something along the lines of:

  ```
  \newcommand*{\mycut}{%
    0.1\textwidth, 0.3\textwidth,
    0.2\textwidth, 0.4\textwidth,
    0.3\textwidth, 0.5\textwidth}
  \begin{shapedcutout}{2}{3}{\mycut}
  ...
  ```

  which is what I used to create the shaped cutout above.
- For an open cutout each entry is the text length of a line. There must be exactly ⟨*numcut*⟩ entries. For instance, given the above definition of `\mycut`

then a call out for an open window would be like:
`\begin{shapedcutout}{2}{6}{\mycut}`

`\picinwindow`

In a shaped cutout the macro `\picinwindow` is placed at the center of the gap in the first line of the cutout. The default `\picinwindow` is a zero-sized picture whose contents is `\putstuffinpic`.

`\picinwindow` and `\putstuffinpic` are initially defined as

```
\newcommand*{\picinwindow}{%
  \begin{picture}(0,0)
    \putstuffinpic
  \end{picture}}
\newcommand*{\putstuffinpic}{}
```

You can change `\putstuffinpic` to place what you want in the picture. For example, to put the large $ symbol in the shaped cutout paragraph above I used:

```
\renewcommand*{\putstuffinpic}{%
  \put(0,-8){\makebox(0,0){\Huge\bfseries \$}}}
```

You have to adjust the placement to suit your purposes and the shape of the cutout.

3 The package code

To try and avoid name clashes, all the internal commands include the string `c@tw`.

3.1 Preliminaries

Announce the name and version of the package, which requires LaTeX 2$_\varepsilon$.

```
⟨*pkg⟩
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{cutwin}[2021/10/13 v0.2 cutout windows]
```

3.2 General

`\c@twwinlines`
`\c@twtoplines`
`\c@twcnt`

We need lots of variables. First some counts.

```
\newcount\c@twwinlines % window lines
\newcount\c@twtoplines % top lines
\newcount\c@twcnt      % a count
```

`\c@twlftside`
`\c@twrtside`
`\c@twtopht`
`\c@twvsilg`

And some lengths.

```
\newdimen\c@twlftside  % left width
\newdimen\c@twrtside   % right width
\newdimen\c@twtopht    % height of top text
\newdimen\c@twvsilg    % Vertical Shift or InterLine Glue
```

`\c@twtoka`
`\c@twtokb`

And some tokens.

```
\newtoks\c@twtoka      % build of parshape spec
\newtoks\c@twtokb      % build of parshape spec
```

`\c@twrawtext`
`\c@twholdwin`
`\c@twwindow`
`\c@twfinaltext`
`\c@twslicea`
`\c@twsliceb`

And some boxes.

```
\newbox\c@twrawtext    % text as input
\newbox\c@twholdwin    % text for window sides
\newbox\c@twwindow     % composed window
\newbox\c@twfinaltext  % final assembled cutout paragraph
\newbox\c@twslicea     % slice A of window text
\newbox\c@twsliceb     % slice B of window text
```

`\c@twdima`
`\c@twdimb`

And some lengths.

```
\newdimen\c@twdima    % formally \@tempdimb
\newdimen\c@twdimb    % formally \@tempdimc
```

`\opencutleft`
`\opencutright`
`\opencutcenter`
`\c@twl@c`

User commands for positioning a cutout; left, right, or center. The default is `\opencutcenter`. `\c@twl@c` is the internal representation.

```
\newcommand*{\opencutleft}{\def\c@twl@c{-1}}
\newcommand*{\opencutright}{\def\c@twl@c{1}}
\newcommand*{\opencutcenter}{\def\c@twl@c{0}}
```

```
\opencutcenter
```

`\cutfuzz` An attempt to stop TeX moaning about over/under full h/v boxes.

```
\newcommand{\cutfuzz}{\vbadness=50000
   \hbadness=50000
%   \hfuzz=1pt
  \sloppy}
```

`\c@twcalcilg` Calculate the interline glue.

```
\newcommand*{\c@twcalcilg}{%
  \c@twvsilg=\baselineskip
  \setbox0=\hbox{(}%
  \advance\c@twvsilg-\ht0 \advance\c@twvsilg-\dp0}
```

3.3 Rectangular cutouts

`\pageinwindow` `\windowpagestuff` User modifiable macros for putting (`\pageinwindow`), via a zero-sized picture, stuff (`\windowpagestuff`) in a cutout window.

```
\newcommand*{\pageinwindow}{%
  \c@twdimb=\c@twwinlines\baselineskip % cutout height
  \c@twdima=\hsize
  \ifnum\c@twl@c=\m@ne % openleft
    \advance\c@twdima -\c@twrtside
  \else
    \ifnum\c@twl@c=\@ne % openright
      \advance\c@twdima -\c@twlftside
    \else% center
      \advance\c@twdima - \c@twlftside
      \advance\c@twdima - \c@twrtside
    \fi
  \fi
  \begin{picture}(0,0)%
    \put(0,0){%
      \raisebox{4pt}{%
%\fbox{%
        \begin{minipage}[t][\c@twdimb][c]{\c@twdima}%
          \windowpagestuff
        \end{minipage}%
%}% end fbox
      }% end raisebox
    }% end put
  \end{picture}}
\newcommand*{\windowpagestuff}{}
```

`cutout` The environment for cutting a rectangular window from a paragraph.

`\begin{cutout}{`⟨*numtop*⟩`}{`⟨*leftwidth*⟩`}{`⟨*rightwidth*⟩`}{`⟨*numcut*⟩`}`

where ⟨*numtop*⟩ is the number of full lines above the window, ⟨*leftwidth*⟩ and ⟨*rightwidth*⟩ are the widths of the text at the sides of the window, and ⟨*numcut*⟩ is the number of lines to be cut (giving the height of the window).

The basic method is to split the paragraph into three parts (a) the top lines above the window, (b) the window lines and (c) the rest (which will be below the window). `\parshape` is used to do the splitting. The top lines are left at their natural length, each line crossing the window is treated as a pair of short lines, and the rest are left at their natural length.

The top lines are put into one box, the windowed ones into another and then there are the remainder. When being boxed, the window lines are combined pairwise to make single lines with space in the middle. Finally, the boxes are output.

```
\newenvironment{cutout}[4]{%
  \cutfuzz
  \c@twtoplines=#1\relax
  \c@twwinlines=#4\relax
  \c@twlftside=#2\relax
  \c@twrtside=#3\relax
  \c@twtoka={}%
```

Generate the `\parshape` specification.

```
  \c@twmakeparspec
```

Reset the arguments and calculate a vertical shift.

```
  \c@twtoplines=#1\relax
  \c@twwinlines=#4\relax
  \c@twcalcshift \vskip-\c@twvsilg
```

Open the `\c@twrawtext` box, call the `\parshape` and start collecting the text to be windowed.

```
  \setbox\c@twrawtext=\vbox\bgroup
  \parshape=\c@twcnt \the\c@twtoka}%
```

Now the code for the actions at `\end{cutout}`, which starts by ending the `\c@rawtext` box, resetting `\parshape` and calculating the interline glue.

```
  {\egroup% end \box\c@twrawtext
  \parshape=0 % reset parshape;
  \c@twcalcilg % find ILG using current font
```

If there are lines above the window, split them off from `\c@twrawtext` into `\c@twfinaltext`.

```
  \ifnum\c@twtoplines>\z@
    \setbox\c@twfinaltext=\vsplit\c@twrawtext to\c@twtoplines\baselineskip
  \fi
```

Calculate the 'height' of the lines that make up the window. If the window is in the center then this is twice the expected height (at this point each final window line is stored as a pair of lines), otherwise it is the expected height based on ⟨*numcut*⟩.

```
  \c@twtopht=\c@twwinlines\baselineskip
  \ifnum\c@twl@c=\z@ % center
    \c@twtopht=2\c@twtopht
  \fi
```

Split the window lines from what is left in the `\c@twrawtext` box into box `\c@twholdwin` which will then contain the narrowed text for the window side(s).

```
  \setbox\c@twholdwin=\vsplit\c@twrawtext to\c@twtopht
```

Now 'compose' the window side(s) text (`\c@twholdwin`) into the final set of windowed lines (`\c@twwindow`). The process depends on whether the cutout is at the left, right, or center.

```
  \ifnum\c@twl@c=\z@% center
    \c@twcompctr{\c@twholdwin}{\c@twwindow}%
  \else% left or right
    \c@twcomplftrt{\c@twholdwin}{\c@twwindow}%
  \fi
```

Assemble the various boxes into the final box (`\c@twfinaltext`) to be output.

```
  \setbox\c@twfinaltext=
    \vbox{\ifnum\c@twtoplines>\z@\unvbox\c@twfinaltext\vskip\c@twvsilg\fi
    \unvbox\c@twwindow%
    \vskip-\c@twvsilg\unvbox\c@twrawtext}%
```

We're done, hand off the paragraph.

```
  \box\c@twfinaltext}
```

`\c@twcompctr`
`\c@twfirst`

`\c@twcompctr{`⟨*linepairbox*⟩`}{`⟨*composedbox*⟩`}` composes a center window box ⟨*linepairbox*⟩ consisting of pairs of short lines into a box ⟨*composedbox*⟩ where the pairs have been assembled into single lines.

`\c@twfirst` is used as a flag for indicating the first line of a cutout.

```
\newcommand*{\c@twcompctr}[2]{%
  \def\c@twfirst{1}%
  \loop\advance\c@twwinlines\m@ne
```

Get a pair of lines and remove skips.

```
  \setbox\c@twslicea=\vsplit#1 to\baselineskip
  \setbox\c@twsliceb=\vsplit#1 to\baselineskip
  \c@twprune{\c@twslicea}{\c@twlftside}%
  \c@twprune{\c@twsliceb}{\c@twrtside}%
```

```
  \ifnum\c@twfirst=\@ne
```

The first time put the texts into a box at the left and right with `\pageinwindow` at the end of the left text.

```
    \setbox#2=\vbox{\unvbox#2\hbox
    to\hsize{\box\c@twslicea\pageinwindow\hfil\box\c@twsliceb}}%
  \else
```

For further lines just put the texts at the left and right.

```
    \setbox#2=\vbox{\unvbox#2\hbox
    to\hsize{\box\c@twslicea\hfil\box\c@twsliceb}}%
  \fi
```

```
  \def\c@twfirst{2}%
  \ifnum\c@twwinlines>\z@\repeat}
```

`\c@twcomplftrt`

Compose an open (left or right) sided rectangular window.

```
\newcommand*{\c@twcomplftrt}[2]{%
  \def\c@twfirst{1}%
  \loop\advance\c@twwinlines\m@ne
```

For an open window we simply deal with one line at a time, not pairs. In other respects the code is generally similar to that for `\c@twcompctr`.

```
  \setbox\c@twslicea=\vsplit#1 to\baselineskip
  \ifnum\c@twl@c=\m@ne%    open left, text at right
    \c@twprune{\c@twslicea}{\c@twrtside}%
    \ifnum\c@twfirst=\@ne
      \setbox#2=\vbox{\unvbox#2\hbox
      to\hsize{\pageinwindow\hfil\box\c@twslicea}}%
    \else
      \setbox#2=\vbox{\unvbox#2\hbox
      to\hsize{\mbox{}\hfil\box\c@twslicea}}%
    \fi
    \def\c@twfirst{2}%
  \else
    \ifnum\c@twl@c=\@ne% open right, text at left
      \c@twprune{\c@twslicea}{\c@twlftside}%
      \ifnum\c@twfirst=\@ne\relax
        \setbox#2=\vbox{\unvbox#2\hbox
        to\hsize{\box\c@twslicea\pageinwindow}}%
      \else
        \setbox#2=\vbox{\unvbox#2\hbox
        to\hsize{\box\c@twslicea}}%
      \fi
      \def\c@twfirst{2}%
    \fi
  \fi
  \ifnum\c@twwinlines>\z@\repeat}
```

`\c@twprune`

`\c@twprune{`⟨*vbox*⟩`}{`⟨*width*⟩`}` chops off the `\lastskip`. It takes a ⟨*vbox*⟩ containing a single `\hbox`, `\unvbox`es it,cancels the `\lastskip` which can be put at the right of a short `\parshape` line, then puts it in a box width ⟨*width*⟩.

```
\newcommand*{\c@twprune}[2]{%
  \unvbox#1 \setbox#1=\lastbox % \box#1 is now an \hbox
  \setbox#1=\hbox to#2{\strut\unhbox#1\unskip}}
```

`\c@twmakeparspec`

Calculate the required `\parshape` spec for a paragraph with a rectangular cutout window.

```
\newcommand*{\c@twmakeparspec}{%
```

`\c@twcnt` is the total number of lines for the `\parshape`, i.e., the number of the top lines plus (twice) the number of window line plus one for the remaining lines.

```
  \c@twcnt=\c@twwinlines
  \ifnum\c@twl@c=\z@
    \multiply \c@twcnt by \tw@
  \fi
  \advance\c@twcnt by \c@twtoplines \advance\c@twcnt by \@ne
```

If there are top lines generate a `0pt \hsize` for each

```
  \ifnum\c@twtoplines>\z@
    \loop\c@twtoka=\expandafter{\the\c@twtoka 0pt \hsize}%
```

```
      \advance\c@twtoplines -1\relax
    \ifnum\c@twtoplines>\z@\repeat
  \fi
```

Now do the cutout portion of the spec.

```
  \ifnum\c@twl@c=\m@ne % openleft
```

For open left calculate the width of the open cutout as `\c@twlftside`.

```
    \c@twlftside=\hsize
    \advance\c@twlftside -\c@twrtside
  \fi
```

Loop over the windowed lines.

```
  \loop\c@twtoka=%
    \ifnum\c@twl@c=\m@ne % openleft
```

For open left generate a `\c@twlftside \c@twrtside` for each.

```
      \expandafter{\the\c@twtoka \c@twlftside \c@twrtside}%
    \else
      \ifnum\c@twl@c=\@ne % openright
```

For open right generate a `\0pt c@twlftside` for each

```
        \expandafter{\the\c@twtoka 0pt \c@twlftside}%
      \else %center
```

For centered generate `0pt \c@twlftside 0pt \c@twrtside` for each pair.

```
        \expandafter{\the\c@twtoka 0pt \c@twlftside 0pt \c@twrtside}%
      \fi
    \fi
    \advance\c@twwinlines \m@ne
  \ifnum\c@twwinlines>\z@\repeat
```

That finishes the cutout portion. For the remaining lines in the paragraph just generate a single `0pt \hsize`.

```
  \c@twtoka=\expandafter{\the\c@twtoka 0pt \hsize}}
```

`\c@twcalcshift` Calculate the estimated vertical shift needed for the window. I determined the values experimentally based on a 10pt font. In may be different for different fonts, but I hope not.

```
\newcommand*{\c@twcalcshift}{% vertical shift
  \c@twvsilg=\c@twwinlines\baselineskip
  \ifnum\c@twtoplines<\@ne
    \advance\c@twvsilg -0.25\baselineskip
  \fi
  \c@twvsilg=0.5\c@twvsilg
  \ifnum\c@twl@c=\z@\else
    \c@twvsilg=0.5\c@twvsilg
  \fi}
```

3.4 Shaped cutouts

`\picinwindow` A zero-sized picture, with contents `\putstuffinpic`, which is placed in the center of the first gap in a shaped cutout.

```
\newcommand*{\picinwindow}{%
  \begin{picture}(0,0)
  \putstuffinpic
  \end{picture}}
```

`\putstuffinpic` Default `\putstuffinpic` is empty.

```
\newcommand*{\putstuffinpic}{}
```

`shapedcutout` `\c@twb` A shaped cutout where the user defines the shape.
`\begin{shapedcutout}{`⟨*numtop*⟩`}{`⟨*numcut*⟩`}{`⟨*shapespec*⟩`}`
where ⟨*numtop*⟩ is the number of full lines above the window, ⟨*numcut*⟩ is the number of lines to be cut (giving the height of the window) and ⟨*shapespec*⟩ is the user's specification of the shape of the surroundings of the cutout. This is in the form of a comma-separated list of either the pairs of widths of the left and right texts of a centered cutout or the widths of the left or right texts of an open cutout.

`\c@twb` holds arg 3 (⟨*shapespec*⟩), the user's parspec.

The code is very similar to that for the `cutout` environment.

```
\newenvironment{shapedcutout}[3]{%
  \cutfuzz
  \c@twtoplines=#1\relax
  \c@twwinlines=#2\relax
  \edef\c@twb{#3}%      user's parspec
```

Generate the top lines portion of the parspec followed by the cutout portion.

```
  \c@twmaketopoddspec
  \c@twbuildoddspec{#3}
```

Continue like the `cutout` code.

```
  \c@twtoplines=#1\relax
  \c@twwinlines=#2\relax
  \c@twcalcshift \vskip-\c@twvsilg
  \setbox\c@twrawtext=\vbox\bgroup
```

`\c@twcnt` is the total number of parshape lines; `\c@twtoka` is the spec for the top lines; `\c@twtokb` is the spec for the cutout lines; and `0pt \hsize` is the spec for the remainder of the paragraph.

```
  \parshape=\c@twcnt \the\c@twtoka \the\c@twtokb 0pt \hsize}%
```

The code for the end of the environment, where most of the work is done. It is similar to the code for the end of the `cutout` environment.

```
 {\egroup
  \parshape=0
  \c@twcalcilg
  \ifnum\c@twtoplines>\z@
    \setbox\c@twfinaltext=\vsplit\c@twrawtext to\c@twtoplines\baselineskip
  \fi
  \c@twtopht=\c@twwinlines\baselineskip
  \ifnum\c@twl@c=\z@ % center
    \c@twtopht=2\c@twtopht
  \fi
  \setbox\c@twholdwin=\vsplit\c@twrawtext to\c@twtopht
  \ifnum\c@twl@c=\z@%    center
    \c@twcompoddctr{\c@twholdwin}{\c@twwindow}%
  \else% open left or right
    \c@twcompoddlftrt{\c@twholdwin}{\c@twwindow}%
  \fi
  \setbox\c@twfinaltext=
    \vbox{\ifnum\c@twtoplines>\z@\unvbox\c@twfinaltext\vskip\c@twvsilg\fi
    \unvbox\c@twwindow%
    \vskip-\c@twvsilg\unvbox\c@twrawtext}%
  \box\c@twfinaltext}
```

`\c@twmaketopoddspec` Make up the easy part of the odd `\parshape` specification; total number `\c@twcnt` and the toplines spec (`\c@twtoka`).

```
\newcommand*{\c@twmaketopoddspec}{%
  \c@twcnt=\c@twwinlines
  \ifnum\c@twl@c=\z@
    \multiply \c@twcnt by \tw@
  \fi
  \advance\c@twcnt by \c@twtoplines \advance\c@twcnt by \@ne
%% \c@twcnt is total of toplines + 2(window lines) + 1
  \c@twtoka={}%
  \ifnum\c@twtoplines>\z@
    \loop\c@twtoka=\expandafter{\the\c@twtoka 0pt \hsize}%
      \advance\c@twtoplines -1\relax
      \ifnum\c@twtoplines>\z@\repeat
  \fi}
```

`\c@twaddtospec` Adds a ‘zero-indented line’ to a parshape spec being assembled in `\c@twtokb`.

```
\newcommand*{\c@twaddtospec}[1]{%
  \c@twtokb=\c@twxpf{\the\c@twtokb 0pt #1 }}
```

`\c@twbuildoddspec` `\c@twxpf` `\c@twbuildoddspec{`⟨*commalist*⟩`}` builds up the parshape spec for the odd cutout lines from the comma-separated list of lengths in ⟨*commalist*⟩.

`\c@twxpf` is a shorthand for `\expandafter` to try and make the code shorter to read.

`\c@twlspec` `\c@twlspec` is used as a temporary variable when iterating over a comma-separated list.

```
\let\c@twxpf\expandafter
\newcommand*{\c@twbuildoddspec}[1]{%
  \c@twtokb={}%
  \@for\c@twlspec:=#1\do{%
    \c@twxpf\c@twxpf\c@twxpf\c@twaddtospec\c@twxpf{\c@twlspec}}}
```

`\c@twcompoddctr`
`\c@twrounds`

Compose the lines of an odd shaped center cutout.

We go through the user's shape list an item at a time but we need to collect pairs of items. The `\c@twrounds` variable is for managing the pairing. `\c@twfirst` is a flag for positioning the `\picinwindow` in the first line of the cutout.

```
\newcommand*{\c@twcompoddctr}[2]{%
  \def\c@twrounds{1}%
  \def\c@twfirst{1}%
  \@for\c@twlspec:=\c@twb\do{%
    \ifnum\c@twrounds=1
      \setbox\c@twslicea=\vsplit#1 to\baselineskip % first of pair
      \c@twprune{\c@twslicea}{\c@twlspec}%
      \def\c@twrounds{2}%
    \else
      \setbox\c@twsliceb=\vsplit#1 to\baselineskip % second of pair
      \c@twprune{\c@twsliceb}{\c@twlspec}%
      \ifnum\c@twfirst=1
        \setbox#2=\vbox{\unvbox#2\hbox
        to\hsize{\box\c@twslicea\hfil\picinwindow\hfil\box\c@twsliceb}}%
        \def\c@twfirst{2}%
      \else
        \setbox#2=\vbox{\unvbox#2\hbox
        to\hsize{\box\c@twslicea\hfil\box\c@twsliceb}}%
      \fi
      \def\c@twrounds{1}%
    \fi}}
```

`\c@twcompoddlftrt`

Compose the open (left or right) lines of an odd shaped cutout.

```
\newcommand*{\c@twcompoddlftrt}[2]{%
  \def\c@twfirst{1}%
  \@for\c@twlspec:=\c@twb\do{%
    \setbox\c@twslicea=\vsplit#1 to\baselineskip % get a line
    \c@twprune{\c@twslicea}{\c@twlspec}%
    \ifnum\c@twl@c=\m@ne%   open left, text at right
      \ifnum\c@twfirst=1
        \setbox#2=\vbox{\unvbox#2\hbox
        to\hsize{\mbox{}\hfil\picinwindow\hfil\box\c@twslicea}}%
        \def\c@twfirst{2}%
      \else
        \setbox#2=\vbox{\unvbox#2\hbox
        to\hsize{\mbox{}\hfil\box\c@twslicea}}%
      \fi
    \else
      \ifnum\c@twl@c=\@ne%  open right, text at left
        \ifnum\c@twfirst=1
        \setbox#2=\vbox{\unvbox#2\hbox
        to\hsize{\box\c@twslicea\hfil\picinwindow\hfil}}%
        \def\c@twfirst{2}%
      \else
        \setbox#2=\vbox{\unvbox#2\hbox
      to\hsize{\box\c@twslicea\hfil}}%
    \fi
  \fi
\fi}}
```

The end of this package.

```
⟨/pkg⟩
```

References

[GM04] Frank Mittelbach and Michel Goossens. *The LaTeX Companion.* Second edition. Addison-Wesley Publishing Company, 2004.

[Hoe87] Alan Hoenig. TeX does windows — The conclusion, *TUGboat*, vol 8, no 2, pp 211–215, 1987.

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.