The roundrect Macros, v2.2

Donald P. Goodman III

February 19, 2016

Abstract
The roundrect macros for METAPOST provide extremely configurable, extremely versatile rectangles (including rounded corners), intended primarily for inclusion in documents produced by TEX and friends. The idea was to provide a METAPOST-based replacement for the incredibly versatile tcolorbox package; the macros are far from achieving that goal. But they are nevertheless extremely useful.

Contents
1 Introduction 1
2 Prerequisites and Conventions 2
3 Shapes and Styles 2
4 Coloring the Parts 4
5 Drop Shadows 6
6 Including Text 8
7 Using External Packages in Text 9
8 Implementation 7

1 Introduction
While TiKZ and its many accompanying packages, particularly tcolorbox, are wonderful and powerful tools, whenever using them I inevitably feel completely lost, and I exert great effort doing comparatively simple things. Contrariwise, thanks to my experience with the drm and dozenal packages, writing in METAPOST is quite straightforward for me. So I decided to try to write some generalized macros to provide functionality similar to that of tcolorbox. It’s not even close
to that kind of flexibility or power, but it’s still quite useful and versatile, so I
make it available for anyone who might be interested.

This document was typeset in accordance with the docstrip utility, which
allows the automatic extraction of code and documentation from the same docu-
ment.

2 Prerequisites and Conventions

Some prerequisites for using this package are METAPOST itself (obviously). If
you’re using the package with \LaTeX, the gmp package would probably be helpful;
be sure to use the latex package option. Finally, the package internally calls
\TeX.mp, so that is also required. All of these should be packaged in any reasonably
modern \LaTeX system, such as \TeXLive or Mik\TeX.

This documentation assumes nothing about your personal \TeX or METAPOST
environment. Con\TeXt and the various forms of Lua\TeX have METAPOST built-
in; with pdf\TeX, the author’s choice, one can use the gmp package to include the
source directly in one’s document (that’s what’s been done in this documentation)
or develop a simple script to compile them afterwards and include them in the
source via \includegraphics (probably the quickest option, since compilation is
done in advance). Here, we simply post the plain vanilla METAPOST code, and
let you work out those details however you prefer.

3 Shapes and Styles

The core of all the action is the roundrect macro; this will set up your rounded
rectangle in the plainest way possible. The first argument is the box’s height, the
second its width, and the third its name, by which you will draw it later:

\begin{verbatim}
roundrect(1in,2in)(rectangle);
draw rectangle;
\end{verbatim}

All the corners don’t have to be rounded; we can make them square if we
want. To do things like this, we use the macro rrborderrad(), which takes a
single argument giving the border radius we want; that is, how rounded we want
the corners of our rectangle. Higher values will be more rounded, lower values will be less:

```
rrborderrad(10pt);
roundrect(1in,2in)(rectangle);
draw rectangle;
```

Notice that the corners in this, with `rrborderrad()` set to 10pt, are much less rounded than the previous example. The default border radius is 40pt, which is quite rounded.

`rrborderrad()` provides an easy way to set the border radius of all four corners at once; however, we can also control each corner individually, with `rrtoplftborderrad`, `rrbotlftborderrad`, `rrtoprtborderrad`, and `rrboottborderrad`, which are parameters rather than macros; that is, we define them using `:=` rather than as an argument in parentheses:

```
rrtoplftborderrad := 20pt;
rrbotlftborderrad := 40pt;
rrtoprtborderrad := 10pt;
rrboottborderrad := 60pt;
roundrect(1in,2in)(rectangle);
draw rectangle;
```

As you can see, this makes it possible to create a large variety of shapes, including the ability to arbitrarily flatten any side of the rectangle desired just by setting the border radius of the appropriate corners to 0pt:
Here, we’ve flattened the top border by setting the top right and top left corners’ border radii to 0pt. This ability to flatten any given side of the rectangle makes it much easier to combine multiple rectangles into interesting forms, which we’ll see a bit more about later.

4 Coloring the Parts

The colors of the roundrect are extremely configurable, both on the whole and for individual parts. The background color of the roundrect is controlled by \texttt{rrinnercolor}, while the border is colored by \texttt{rrbordercolor}.

\begin{verbatim}
rrbordercolor(blue);
rrinnercolor := red;
roundrect(1in,2in)(rectangle);
draw rectangle;
\end{verbatim}

By default, \texttt{rrinnercolor} is white and \texttt{rrbordercolor} is black. Notice that \texttt{rrbordercolor} is a macro, not a parameter; that’s because each border can be individually colored, and this macro simply does all of them at once. We’ll see more about that later.

You can also completely suppress the border by using \texttt{rrnotop}, \texttt{rrnobot}, \texttt{rrnolft}, and \texttt{rrnort}, which is particularly useful when you want to combine multiple rectangles without making an obvious border between them. You can combine these in any way you like:
Here we’ve squared all the corners to make it easier to see what’s going on. Each border can be colored individually and separately from the others, using the commands you’d expect:

```latex
rrbordercolor(blue);
rrinnercolor := red;
rrnotop := true;
rrnobot := true;
rrborderrad(0pt);
roundrect(1in,2in)(rectangle);
draw rectangle;
```

There is obviously some difficulty in determining what part of each rounded corner should be colored how; this ability is typically more useful with a single, flattened side, to help it blend in better when combined with other constructs:

```latex
rrtopbordercolor := blue;
rrbotbordercolor := green;
rrlftbordercolor := red;
rrrtbordercolor := black;
rrborderrad(20pt);
roundrect(1in,2in)(rectangle);
draw rectangle;
```

```latex
rrbordercolor(black);
rrbotbordercolor := green;
rrinnercolor := red;
rrborderrad(20pt);
rrbotlftborderrad := 0pt;
rrbotrtborderrad := 0pt;
roundrect(1in,2in)(rectangle);
draw rectangle;
```
Perhaps you don’t like the border; you’d like it thicker, or drawn with a square rather than a circular pen. You’re in luck; \texttt{rrborderpen()} takes the single argument of the pen you’d like to draw the border with, defined like any other \texttt{METAPOST} pen:

```
rrborderpen(pensquare scaled 3);
roundrect(1in,2in)(rectangle);
draw rectangle;
```

The default border pen is \texttt{pencircle scaled 1.5}, so this results in a square pen rather than a circular one, twice as thick. You can also use individual pens for each border, as expected:

```
rrbotlftborderrad := 0pt;
rrbotrborderrad := 0pt;
rrbotbordercolor := green;
rrbotborderpen := pensquare
    yscaled 6;
roundrect(1in,2in)(rectangle);
draw rectangle;
```

Here we’ve flattened the bottom border, colored it green, and drawn it with a square pen scaled on the y-axis only by 6. Clearly, there are huge possibilities here.

5 Drop Shadows

We can also put a \texttt{shadow} on the boxes using \texttt{rrdropshadow}, a boolean value which defaults to \texttt{false}:
The drop shadow always mimics the shape of the box itself; there is presently no way to avoid that. If for some reason you want to, you'll have to create a separate `roundrect` and place it manually.

We can control the size and direction of the drop shadow fairly easily, however, along with its color. Its color is controlled by `rrshadowcolor`, which can be set to any arbitrary METAPOST color:

```
rrdropshadow := true;
rrshadowcolor := blue;
roundrect(1in,2in)(rectangle);
draw rectangle;
```

The position of the drop shadow is governed by `rrshadowx` and `rrshadowy`, which will shift the `roundrect` on the x or y axis, respectively. By default, these are set to one quarter of the border radius in effect for the bottom left corner:
6 Including Text

Finally, we can put text in the rectangles; this is as configurable as everything else:

```
rrbodytext := "Let’s put some
text into this rectangle and
see if it typesets
correctly!";
roundrect(1in,2in)(rectangle);
draw rectangle;
```

The font and style of the text can be controlled with `rrtextfont`, and the color of the text can be controlled with `rrtextcolor`: 

```
rrtextfont := true;
rrshadowcolor := blue;
rrshadowx :=
  -rrbotlftborderrad/4;
rrshadowy := rrbotlftborderrad/4;
roundrect(1in,2in)(rectangle);
draw rectangle;
```
We also used, without explaining it first, `rrtextalign`, which allows insertion of text alignment commands. This can also be inserted in the `rrtextfont` variable, but it seemed logical to have a separate parameter for it. It’s default is `\centering`.

The width of the text is governed by `rrtextwd`, which defaults to the same width as the rectangle with a 3pt buffer on either side. The buffer is not directly controllable, but the width can be set however you like:

Finally, to restore all these values to the default, use the `rrrestorevals;` directive. This will clear everything to default so you can have a completely different `roundrect` in the same figure.

### 7 Using External Packages in Text

Frequently, of course, the `rrtextfont` options will be either insufficient or overly cumbersome for your needs. For example, you might want all the text in your labels to be in a different font; to match your main body font, for example.

`roundrect` offers `rrusepackage` for this purpose. It is a string, designed specifically for the purpose of including arbitrary `\LaTeX` packages for typesetting text.
For example, if your main body font is EB Garamond, the easiest way to get your text to match that is to ask \texttt{METAPOST} to use the \texttt{ebgaramond} package when it typesets:

\begin{verbatim}
rrusepackage := "\\usepackage\{ebgaramond\}";
rrbodytext := "This is \texttt{EB} Garamond, not Computer Modern."
roundrect(1in,2in)(rectangle);
draw rectangle;
\end{verbatim}

This is \texttt{EB} Garamond, not Computer Modern.

Notice that \texttt{rrusepackage := "\\usepackage\{ebgaramond\}";} takes care of changing the body font \textit{and} of defining the \texttt{\textsw} environment (itself defined in \texttt{ebgaramond}), which we would otherwise have to do separately.

To switch this back, simply reset the string to empty:

\begin{verbatim}
rrusepackage := "";
rrbodytext := "This is Computer Modern, not EB Garamond."
roundrect(1in,2in)(rectangle);
draw rectangle;
\end{verbatim}

This is Computer Modern, not EB Garamond.

Using color commands (e.g., from \texttt{color} or \texttt{xcolor}) will not throw errors, provided the appropriate package is included; however, it will not work. This seems to be an unavoidable consequence of the way that \texttt{TEX.mp} works; \texttt{TEX} \texttt{\special} commands are destroyed in the process, and there isn’t really a robust way to do it without this effect.

8 Implementation

\begin{verbatim}
1 input TEX;
2 color rrinnercolor; rrinnercolor := white;
3 numeric rrtoprtborderrad; rrtoprtborderrad := 40pt;
\end{verbatim}
numeric rrbotrtborderrad; rrbotrtborderrad := 40pt;
numeric rrbotlftborderrad; rrbotlftborderrad := 40pt;
numeric rrtoplftborderrad; rrtoplftborderrad := 40pt;
numeric rrtextwd; rrtextwd := 0;
numeric rrshadowx; rrshadowx := rrbotrtborderrad/4;
numeric rrshadowy; rrshadowy := -rrbotrtborderrad/4;
string rrtextfont; rrtextfont := "\fontsize{10pt}{12pt}\selectfont ";
color rrtextcolor; rrtextcolor := black;
string rrbodytext; rrbodytext := "";
string rrtextalign; rrtextalign := "\centering";
string rrusepackage; rrusepackage := "";
boolean rrnotop; rrnotop := false;
boolean rrnobot; rrnobot := false;
boolean rrnolft; rrnolft := false;
boolean rrnort; rrnort := false;
boolean rrdropshadow; rrdropshadow := false;
color rrtopbordercolor; rrtopbordercolor := black;
color rrbotbordercolor; rrbotbordercolor := black;
color rrlftbordercolor; rrlftbordercolor := black;
color rrrtbordercolor; rrrtbordercolor := black;
color rrshadowcolor; rrshadowcolor := black;
def rrbordercolor(expr x) =
  rrtopbordercolor := x;
  rrbotbordercolor := x;
  rrlftbordercolor := x;
  rrrtbordercolor := x;
enddef;
def rrborderrad(expr x) =
  rrtoplftborderrad := x;
  rrbotlftborderrad := x;
  rrtoprtborderrad := x;
  rrbotrtborderrad := x;
enddef;
pen rrtopborderpen; rrtopborderpen := pencircle scaled 1.5;
pen rrbotborderpen; rrbotborderpen := pencircle scaled 1.5;
pen rrlftborderpen; rrlftborderpen := pencircle scaled 1.5;
pen rrrtborderpen; rrrtborderpen := pencircle scaled 1.5;
def rrborderpen(expr x) =
  rrtopborderpen := x;
  rrbotborderpen := x;
  rrlftborderpen := x;
  rrrtborderpen := x;
enddef;
def rrrestorevals =
  rrborderrad(40pt);
  rrbordercolor(black);
  rrborderpen(pencircle scaled 1.5);
  rrinnercolor := white;
  rrnotop := false;
  rrnobot := false;
rrbotrcorner := h{down}..{left}g;
rrbotlftcorner := f{left}..{up}m;
rrtopborder := rrtoplftcorner--rrtoprtcorner;
rrbotborder := rrbotrcorner--rrbotlftcorner;
rrlftborder := rrbotlftcorner--rrtoplftcorner;
rrrtborder := rrtoprtcorner--rrbotrtcorner;
picture rrdropshadowpic;
if (rrdropshadow = true):
rrdropshadowpic := image(fill rrtoplftcorner--rrtoprtcorner--
rrbotrtcorner--rrbotlftcorner--cycle
shifted (rrshadowx,rrshadowy) withcolor
rrshadowcolor);
else:
rrdropshadowpic := currentpicture;
fi
name := currentpicture;
addto name also rrdropshadowpic;
rrdropshadowpic := image(fill rrtoplftcorner--rrtoprtcorner--
rrbotrcorner--rrbotlftcorner--cycle withcolor
rrinnercolor);
addto name also rrdropshadowpic;
% name := image(fill rrtoplftcorner--rrtoprtcorner--
% rrbotrcorner--rrbotlftcorner--cycle withcolor
% rrinnercolor);
picture rrtmpborder;
border := currentpicture;
if (rrnotop = false):
rrtmpborder := image(draw rrtopborder withcolor
rrtopbordercolor withpen rrtopborderpen);
addto border also rrtmpborder;
fi
if (rrnobot = false):
rrtmpborder := image(draw rrbotborder withcolor
rrbotbordercolor withpen rrbotborderpen);
addto border also rrtmpborder;
fi
if (rrnolft = false):
rrtmpborder := image(draw rrlftborder withcolor
rrlftbordercolor withpen rrlftborderpen);
addto border also rrtmpborder;
fi
if (rrnort = false):
rrtmpborder := image(draw rrrtborder withcolor
rrrtbordercolor withpen rrrtborderpen);
addto border also rrtmpborder;
fi
addto name also border;
rrtext :=
image(label(TEX("\parbox{\decimal(rrtextwd)bp}{&rrtextalign&rrtextfont &"&rrbodytext&"}"),
addto name also rrtext;
enddef;