Fonts, Lines, and Transparency in R Graphics

by Paul Murrell

Introduction

For R version 2.0.0, a number of basic graphics facilities have been added to allow finer control of fonts, line styles, and transparency in R graphics. At the user level, these changes involve the addition of graphical parameters in the **graphics** and **grid** (Murrell, 2002) packages for specifying fonts and line styles, and, in the **grDevices** package, some new functions for controlling font specifications and changes to some existing functions for specifying semitransparent colours. Summaries of the changes to the user interface and the availability of the changes on the standard devices are given in Tables 2 and 3 at the end of the article.

Now you can choose your family

The specification of a particular font can be a complex task and it is very platform dependent. For example, the X11 specification for a Courier font is of the form `"-*-courier-%s-%s-*-*-%d-*-*-*-*-*-*-*"` while for Windows the specification is something like `"TT Courier"`.

To make things simpler and more standardised, R graphics has adopted a device-independent mechanism for specifying fonts which is loosely modelled on the specification of fonts in the Cascading Style Sheets specification (CSS, 1998).

User interface

R graphics provides three basic parameters for specifying a font: the font family, the font face, and the font size.

The specification of font size and face have not changed. The font size is controlled both by an absolute pointsize, via `par(ps)` in the **graphics** packages or `gpar(fontsize)` in the **grid** package, and a relative multiplier, `par(cex)` or `gpar(cex)`. The font face is specified in the **graphics** package via `par(face)` as an integer between `1` and `5` and in the **grid** package via `gpar(fontface)` as a string: `"plain"`, `"bold"`, `"italic"` (or `"oblique"`), `"bold.italic"`, or `"symbol"`.

The specification of font families is where the changes have been made. All graphics devices define an initial or default font family when the device is created. This is typically a sans-serif font such as Helvetica or Arial. A new font family is specified via `par(family)` in the **graphics** package or `gpar(fontfamily)` in the **grid** package using a device-independent family name.

Four standard families, `"serif"`, `"sans"`, `"mono"`, and `"symbol"` are provided by default and more may be defined. Devices with support for font families provide a font database which is used to map the device-independent font family to a device-specific font family specification. For example, the standard mappings for the Quartz device are shown below.

```
> quartzFonts()

$serif
[1] "Times-Roman"
[2] "Times-Bold"
[3] "Times-Italic"
[4] "Times-BoldItalic"

$sans
[1] "Helvetica"
[2] "Helvetica-Bold"
[3] "Helvetica-Italic"
[4] "Helvetica-BoldOblique"

$mono
[1] "Courier"
[2] "Courier-Bold"
[3] "Courier-Oblique"
[4] "Courier-BoldOblique"

$symbol
[1] "Symbol" "Symbol" "Symbol"
[4] "Symbol"
```

For each of the standard devices there is a new function of the form `<dev>Font()` for defining new mappings and a new function of the form `<dev>Fonts()` for querying and modifying font family mappings and for assigning new mappings to the font database for the device (see Table 2).

This approach means that it is now possible to modify the graphics font family on all of the core graphics devices (it was only previously possible on Windows), and the font family specification is now consistent across all devices (it is device-independent and the interface for modifying the font family and/or specifying new font families is consistent).

Hershey fonts

It is possible to specify a Hershey vector font (Hershey, 1969) as the font family. These fonts are device-independent and are drawn by R so they are available on all devices. Table 1 lists the Hershey font families that are provided.

Not all families support the standard font faces and there are three non-standard font faces supported by the "HersheySerif" family: face 5 is a Cyrillic font, face 6 is an oblique Cyrillic font, and face 7 provides a set of Japanese characters (for more information, see help(Hershey)).

Device support

All of the standard graphics devices provide device-independent font family mappings, however there are some limitations. For a start, the user is responsible for ensuring that all fonts are available and installed correctly on the relevant system.

The PostScript device also requires that you specify font metric files for the font you are mapping to. Furthermore, all fonts that are to be used on a PostScript device must be "declared" when the device is created (see the new fonts argument to postscript()). Finally, it is not possible to modify a PostScript font family mapping while the mapping is being used on a device.

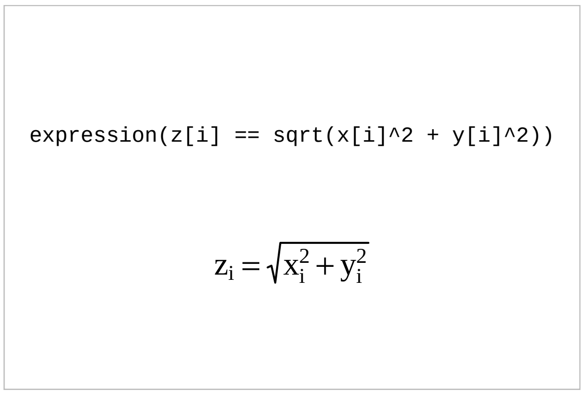
The PDF device uses the PostScript font database (there is neither a pdfFont() nor a pdfFonts() function). Also, the PDF device does not embed fonts in the PDF file, which means that the only font families that can be mapped to are the "base 14" fonts that are assumed to be available in a PDF viewer: "Times" or "Times New Roman", "Helvetica" or "Arial", "Courier", "Symbol", and "ZapfDingbats".[1]

On the Windows graphics device, the font family mapping will override the mapping in the Rdevga file when the font family is not "" and the font face is between 1 ("plain") and 4 ("bold.italic").

An example

The following code produces output demonstrating a mathematical annotation (see Figure 1). The text that shows the code used to produce the annotation is is drawn in a "mono" font and the text that shows what the annotation looks like is drawn with a "serif" font. This example runs unaltered on all of the core graphics devices.

```
> expr <- "z[i] == sqrt(x[i]^2 + y[i]^2)"
> grid.text(paste("expression(",
+     expr, ")", sep = ""),
+     y = 0.66, gp = gpar(fontfamily = "mono",
+         cex = 0.7))
> grid.text(parse(text = expr),
+     y = 0.33, gp = gpar(fontfamily = "serif"))
```



```
expression(z[i] == sqrt(x[i]^2 + y[i]^2))
```

$$z_i = \sqrt{x_i^2 + y_i^2}$$

Figure 1: Simple R graphics output using "mono" and "serif" font families.

The end of the line

The concepts of line ending style and line join style have been added to R graphics.

All lines are drawn using a particular style for line ends and joins, though the difference only becomes obvious when lines become thick. Figure 2 shows an extreme example, where three very wide lines (one black, one dark grey, and one light grey) have been drawn through exactly the same three locations. The locations are shown as black dots.

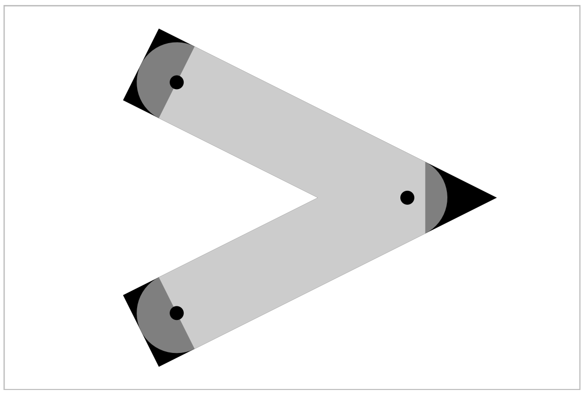


Figure 2: The different line end and line join styles.

[1]The "14" comes from the fact that there are four different faces for each of the Times, Helevetica/Arial, and Courier font families.

Table 1: The Hershey vector fonts available in R.

Family	Description	Supported faces
`"HersheySerif"`	Serif font family	1 to 7
`"HersheySans"`	Sans serif font family	1 to 4
`"HersheyScript"`	Script (handwriting) font family	1 and 2
`"HersheyGothicEnglish"`	Various gothic font families	1
`"HersheyGothicGerman"`		
`"HersheyGothicItalian"`		
`"HersheySymbol"`	Serif symbol font family	1 to 4
`"HersheySansSymbol"`	Sans serif symbol font family	1 and 3

The black line is drawn with `"square"` ends and a `"mitre"` join style, the dark grey line is drawn with `"round"` ends and a `"round"` join style, and the light grey line is drawn with `"butt"` ends and a `"bevel"` join style.

When the line join style is `"mitre"`, the join style will automatically be converted to `"bevel"` if the angle at the join is too small. This is to avoid ridiculously pointy joins. The point at which the automatic conversion occurs is controlled by a mitre limit parameter, which is the ratio of the length of the mitre divided by the line width. The default value is `10` which means that the conversion occurs for joins where the angle is less than 11 degrees. Other standard values are `2`, which means that conversion occurs at angles less than 60 degrees, and `1.414` which means that conversion occurs for angles less than 90 degrees. The minimum mitre limit value is `1`.

It is important to remember that line join styles influence the corners on rectangles and polygons as well as joins in lines.

User interface

The current line end style, line join style, and line mitre limit can be queried and set in the **graphics** package via new `par()` settings: `lend`, `ljoin`, and `lmitre` respectively.

In the **grid** package, the parameters are `"lineend"`, `"linejoin"`, and `"linemitre"`, and they are settable via the `gp` argument of any viewport or graphics function using the `gpar()` function.

Device support

Line end styles and line join styles are not available on the Windows graphics device and it is not possible to control the line mitre limit on the X11 device (it is fixed at 10).

Fine tuning the alpha channel

It is now possible to define colours with a full alpha channel in R.

The alpha channel controls the transparency level of a colour; a colour with an alpha value of 0 is fully transparent and an alpha value of 1 (or 255, depending on the scale being used) means the colour is fully opaque. Anything in between is semitransparent.

User interface

Colours may be specified with an alpha channel using the new `alpha` argument to the `rgb()` and `hsv()` functions. By default opaque colours are created.

It is also possible to specify a colour using a string of the form `"#RRGGBBAA"` where each pair of characters gives a hexadecimal value in the range 0 to 255 and the `AA` pair specify the alpha channel.

The function `col2rgb()` will report the alpha channel for colours if the new `alpha` argument is `TRUE` (even if the colour is opaque). Conversely, it will *not* print the alpha channel, even for semitransparent colours, if the `alpha` argument is `FALSE`.

When colours are printed, anything with an alpha channel of 0 is printed as `"transparent"`. Known (opaque) colours are printed using their R colour name, e.g., `rgb(1, 0, 0)` is printed as `"red"`. Otherwise, opaque colours are printed in the form `"#RRGGBB"` and semitransparent colours are printed as `"#RRGGBBAA"`.

In the **grid** package, there is also an `alpha` graphical parameter (specified via `gpar()`), which controls a general alpha channel setting. This setting is combined with the alpha channel of individual colours by multiplying the two alpha values together (on the $[0, 1]$ scale). For example, if a viewport is pushed with `alpha=0.5` then everything drawn within that viewport will be semitransparent.

Device support

Most graphics devices produce no output whatsoever for any colour that is not fully opaque. Only the PDF and Quartz devices will render semitransparent colour (and, for the PDF device, only when the new `version` argument to `pdf()` is set to `"1.4"` or higher).

An example

A simple example of the application of transparency in a statistical graphic is to provide a representation of the density of data when points overlap. The following code plots 500 random values where each data symbol is drawn with a semitransparent blue colour. Where more points overlap, the blue becomes more saturated (see Figure 3).

```
> pdf("alpha.pdf", version = "1.4",
+     width = 4, height = 4)
> par(mar = c(5, 4, 2, 2))
> plot(rnorm(500), rnorm(500),
+     col = rgb(0, 0, 1, 0.2),
+     pch = 16)
> dev.off()
```

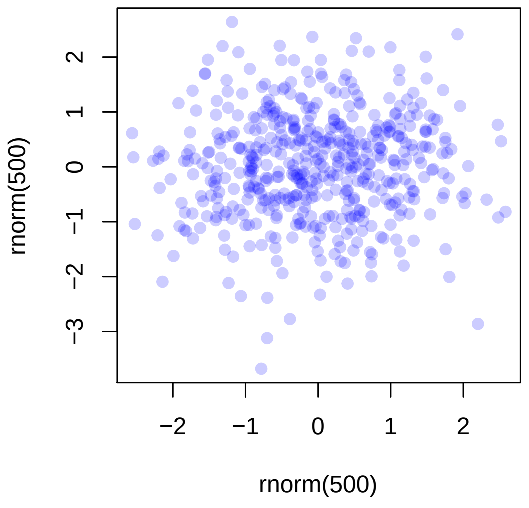


Figure 3: An application of alpha transparency.

Bibliography

CSS2 specification. Technical report, World Wide Web Consortium, http://www.w3.org/TR/1998/REC-CSS2-19980512, May 1998. The latest version of CSS2 is available at http://www.w3.org/TR/REC-CSS2. 5

A. V. Hershey. Fortran IV programming for cartography and typography. Technical Report TR-2339, U. S. Naval Weapons Laboratory, Dahlgren, Virginia, 1969. 6

P. Murrell. The grid graphics package. *R News*, 2(2): 14–19, June 2002. URL `http://CRAN.R-project.org/doc/Rnews/`. 5

Table 2: Summary of the availability of new features on the standard graphics devices. A dot (•) means that the feature is supported on the device.

Feature	PostScript	PDF	X11	Windows	Quartz
Font family	•	•	•	•	•
Line end style	•	•	•		•
Line join style	•	•	•		•
Line mitre limit	•	•			•
Alpha channel		•			•

Table 3: Summary of the new and changed functions in R 2.0.0 relating to fonts, line styles, and transparency.

Package	Function	Description
grDevices	`rgb()`	New `alpha` argument for specifying alpha channel.
	`hsv()`	New `alpha` argument for specifying alpha channel.
	`col2rgb()`	New `alpha` argument for reporting alpha channel.
	`postscriptFont()`	Generates a PostScript-specific font family description.
	`postscriptFonts()`	Defines and reports font mappings used by the PostScript and PDF devices.
	`windowsFont()`	Generates a Windows-specific font family description.
	`windowsFonts()`	Defines and reports font mappings used by the Windows device.
	`quartzFont()`	Generates a Quartz-specific font family description.
	`quartzFonts()`	Defines and reports font mappings used by the Quartz device.
	`X11Font()`	Generates an X11-specific font family description.
	`X11Fonts()`	Defines and reports font mappings used by the X11 device.
graphics	`par()`	New parameters `lend`, `ljoin`, and `lmitre` for controlling line style. New `family` parameter for controlling font family.
grid	`gpar()`	New graphical parameters, `lineend`, `linejoin`, and `linemitre` for controlling line style. The `alpha` parameter now affects the alpha channel of colours. The `family` parameter now affects graphical devices.