Alpha Transparency in R Graphics

Paul Murrell

August 17, 2004

Internally, R stores colours as 32-bit integers. From the start, 24 bits have been used to store three 8-bit channels for Red, Green, and Blue. More recently, the remaining 8 bits have been used to store an alpha channel to represent colour transparency. From version 1.4.0, R graphics has acknowledged an alpha channel of 0 as opaque and anything else as completely transparent. From 2.0.0, the whole range of alpha values (0 to 255) is available.

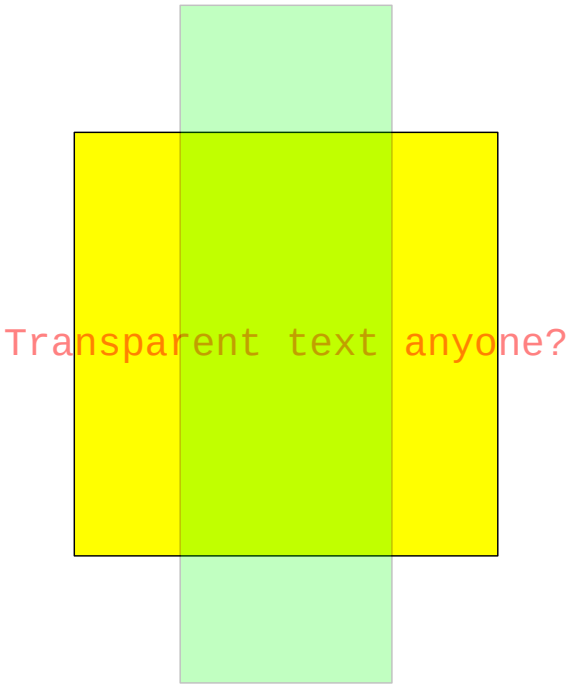
R passes colours to devices and they may or may not implement alpha transparency. Most will implement 0 as opaque and everything else as fully transparent. From 2.0.0, the PDF device implements the whole range of alpha values.

The only way to specify alpha values at present are via the special colour name `"transparent"`, which produces a fully transparent white, or using the `alpha` graphical parameter in `grid` (see the examples below).

A `grid` and PDF example

The `alpha` graphical parameter is a number from 0 to 1 (opaque to transparent). The code below draws an opaque yellow rectangle, then transparent red text overlapping the rectangle, then a transparent green rectangle over the top of both (on a PDF device). On other devices, the text and second rectangle will not be drawn at all.

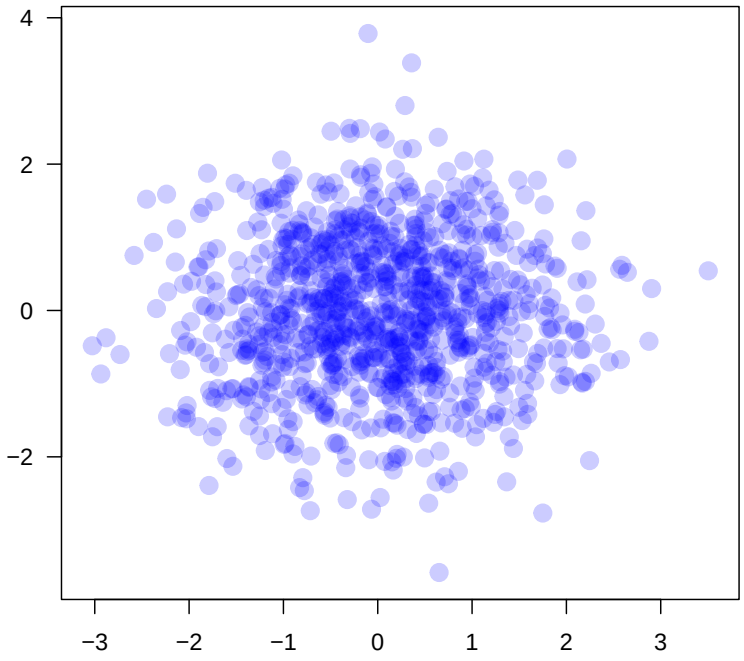
```
> pdf("transfont.pdf", version = "1.4")
> grid.rect(width = 0.5, height = 0.5, gp = gpar(fill = "yellow"))
> grid.text("Transparent text anyone?", gp = gpar(alpha = 0.5,
+     fontsize = 20, fontfamily = "mono", col = "red"))
> grid.rect(width = 0.25, height = 0.8, gp = gpar(fill = "green",
+     alpha = 0.5))
> dev.off()
```



A "real" example

Here is one potentially useful application of transparency in a plot. Points are drawn in a transparent blue colour and the overlapping points produce a visual cue of point density.

```
> pdf("realtrans.pdf", version = "1.4")
> x <- rnorm(1000)
> y <- rnorm(1000)
> pushViewport(plotViewport(c(5, 4, 4, 2)), dataViewport(x,
+     y))
> grid.rect()
> grid.xaxis()
> grid.yaxis()
> grid.points(x, y, pch = 16, gp = gpar(col = "blue",
+     alpha = 0.2))
> popViewport(2)
> dev.off()
```



To Do

The Quartz device should be able to do transparency (see Section "For Device Maintainers" below). I presume that Windows is capable of transparency, but I don't know how hard it would be to get `R`'s Windows device to do it.

There needs to be an R-level user interface for specifying transparent colours. Some obvious ones would be an `rgba()` function and possibly allowing colour names of the form `"#AARRGGBB"`. Conversions from internal colours to R-level colours also need to be implemented.

For Device Maintainers

Most devices I have looked at (X11, PostScript, Windows, GTK, and XFig) currently check for `R_OPAQUE(col)` (or, equivalently, `R_ALPHA(col) == 0`) and only draw if that is true (the PicTeX driver does not use colour). No change is required for such code.

The Quartz device currently checks for `col == NA_INTEGER` and sets `alpha = 0.0` in such cases. This will produce the wrong result now, but should be easily fixable by simply removing the check and setting `alpha = R_ALPHA(col)/255.0` (i.e., implementing alpha

transparency!). In a couple of places there are also some colour initialisations of the form `col = NA_INTEGER`; these should be replaced by `col = R_RGBA(255, 255, 255, 255)`.

For R-core

A missing colour (`col = NA`) used to map to an internal colour of `NA_INTEGER`. The mapping is now to `R_RGBA(255, 255, 255, 255)` (i.e., fully transparent white). This means that there is now no internal representation of `NA` for a colour, which means that any code which wants to check whether a colour is `NA` must do so with the R-level colour and NOT the internal colour.

This also means that `NA` cannot round-trip via the internal colour format. For example ...

```
> par(col = NA)
> par("col")

[1] "transparent"
```

I can't see a way around this.