Writing new grob functions

Hadley Wickham

2005-11-16

As you use `ggplot` more, you may discover some of the limitations of the included grob functions and want to write your own. (Although I'm very open to suggestions and if you develop a useful grob function I will be happy to include in the main package). This document will discuss in detail how grob functions work and what you need to do to build your own. As an example I will work through the process of creating a grob function that produces a 1d density plot with jittered rug plot, not unlike that created by `densityplot` in `lattice`.

`ggplot` uses grid graphics, so to be able to create new grob functions you will need some familiarity with basic grid functions. This isn't too hard to acquire, and I suggest you look at the code for the basic grob functions (eg. `grob_points`, `grob_rect`) for some simple examples.

There are three functions you will need to write when creating a new grob function:

- A convenient function for adding your grob function to a plot. This is what the users uses to add the grob to the plot. This provides a convenient place to modify other plot settings (eg. scales) that your grob might need. See `gghistogram` and `ggpoint` for examples.
- The grob function. This converts a list of aesthetics, plus some optional parameters to a gList of grobs. This function is prefixed with `grob_`.
- The preprocessor function (optional). If your grob function creates new aesthetics (like the densityplot will, by creating a new y position aesthetic) you will need this preprocessing stage so that the new aesthetics are available for the scale functions. This function is prefixed with `pre_`

Example

In this example, we're going to develop a grob function to display a 1d density with jittered grob plot. We first need to decide what aesthetics and what optional parameters this function will take. We want to give the user some flexibility over the how the density is computed, and the appearance of the density plot. So let's use the following option parameters:

- adjust: adjustment to default bandwidth
- kernel: type of kernel to use
- colour: the colour of the density line

There is only one aesthetic we need: x. From this we will compute the density and create a y aesthetic. We do this in the `pre_density` function. Remember that this function returns a data frame that will be used by scales, and then by `grob_density`

```
pre_density <- function(data, adjust=1, kernel="gaussian", ...) {
   dens <- density(data$x, adjust=adjust, kernel=kernel)
   dens$'.type' <- "density"
```

```
   rug <- data.frame(x = jitter(data$x), y=-0.5, .type="rug")
   rbind(as.data.frame(dens[c("x","y",".type")]), rug)
}
```

We can only return one data frame, so we need to package up the data for both the density and rug somehow. I've chosen to do this by adding an extra column to the data frame called `.type` (so named to avoid conflict with user variables) that we will use to determine where the data should go.

The next task is to write grob function to draw the density (with lines) and the jittered rug plot (if necessary).

```
grob_density <- function(aesthetics, colour="black", ...) {
   aesthetics <- data.frame(aesthetics)
   dens <- subset(aesthetics, .type == "density")
   rug <- subset(aesthetics, .type == "rug")

   dens$colour <- colour

   gTree(children = gList(
      grob_line(dens),
      grob_rect(rug, colour="black")
   ))

}
```

Finally, we write a convenience function to make it easy to for users. This function also changes the y label to density, and sets up an appropriate scale.

```
ggdensity <- function(plot = .PLOT, aesthetics=list(), ..., data=plot$data) {
   plot$ylabel <- "Density"
   plot <- pscontinuous(plot, "y", range=c(0,NA), expand=c(0.05,0))
   gg_add("density", plot, aesthetics, ..., data=data)
}
```