

What's in a Name?

Paul Murrell

The University of Auckland

NZSA Conference August 2011

Overview

- Complex statistical graphics consist of lots of individual components.
- Many systems provide control over all aspects of a graphic through a large, nasty list of parameters (e.g., R's `par()` function).
- An alternative approach is to label the individual components of a graphic and provide tools for accessing and modifying them.
- More generally, the ability to post-process a graphic leads to some interesting possibilities.

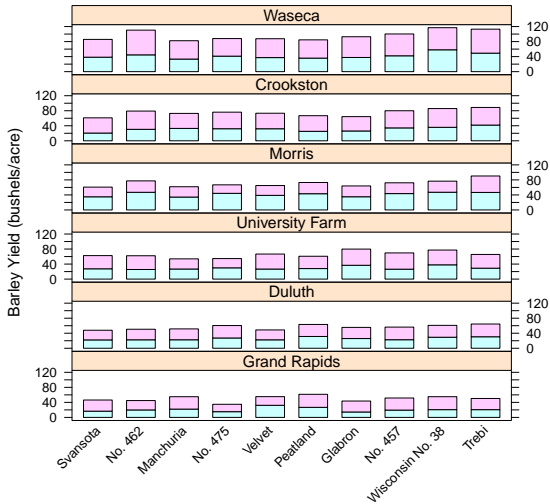
Complex Statistical Graphics

- Complex plots contain **many** individual components.

```
> library(lattice)
```

```
> barchart(yield ~ variety | site, data = barley,  
           groups = year, layout = c(1,6), stack = TRUE,  
           ylab = "Barley Yield (bushels/acre)",  
           scales = list(x = list(rot = 45)))
```

Complex Statistical Graphics



Complex Statistical Graphics

- The interface provided for controlling all of those details often consists of a very large list of parameters.
- In **lattice** this is provided by the `trellis.par.get()` and `trellis.par.set()` functions.

Complex Statistical Graphics

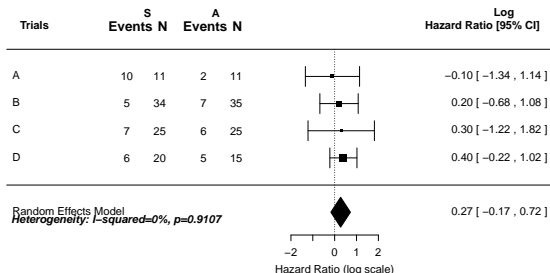
grid.pars	lty lwd	col	lineheight	key.right right.padding
fontsize	box.umbrella	strip.border	axis.components	box.3d
text	alpha	alpha	left	alpha
points	col	col	top	col
background	lty	lty	right	lty
alpha	lwd	lwd	bottom	lwd
col	dot.line	superpose.line	layout.heights	par.xlab.text
panel.background	alpha	alpha	top.padding	alpha
col	col	col	main	cex
clip	col	lty	main.key.padding	col
panel	lty	lwd	key.top	font
strip	lwd	superpose.symbol	xlab.top	lineheight
add.line	dot.symbol	alpha	key.axis.padding	par.ylab.text
alpha	alpha	cex	axis.top	alpha
col	cex	col	strip	cex
lty	col	fill	panel	col
lwd	font	font	axis.panel	font
add.text	pch	pch	between	lineheight
alpha	plot.line	superpose.polygon	axis.bottom	par.zlab.text
cex	alpha	alpha	axis.xlab.padding	alpha
col	col	col	xlab	cex
font	lty	border	xlab.key.padding	col
lineheight	lwd	lty	key.bottom	font
plot.polygon	plot.symbol	lwd	key.sub.padding	lineheight
alpha	alpha	regions	sub	par.main.text
col	cex	alpha	bottom.padding	alpha
border	col	col	layout.widths	cex
lty	font	shade.colors	left.padding	col
lwd	pch	alpha	key.left	font
box.dot	fill	palette	key.ylab.padding	lineheight
alpha	reference.line	axis.line	ylab	par.sub.text
col	alpha	alpha	ylab.axis.padding	alpha
cex	col	col	axis.left	cex
font	lty	lty	axis.panel	col
pch	lwd	lwd	strip.left	font
box.rectangle	strip.background	axis.text	panel	lineheight
alpha	alpha	alpha	between	
col	col	cex	axis.right	
fill	col	col	axis.key.padding	
	strip.shingle	font	ylab.right	
	alpha			

Large lists of parameters or arguments

- There are a number of problems with these sorts of interfaces:
 - It is hard to know/discover which parameter or argument controls which feature of the plot.
 - This is not helped by the fact that the list of parameters or arguments tends to be “flat”, which does not reflect the structure of the graphic.
 - The interface is doomed to be uncomprehensive; it is inevitable that the user will want to change something that is inaccessible via the interface.
 - It is hard or impossible for the user to modify or extend the interface.

Large lists of parameters or arguments

- R-help 2011-24-08 from Paola Tellaroli.



“Even if I have specified 'col="red", border="red"', color of squares and diamond rests black! Why?”

Large lists of parameters or arguments

- R-help 2011-26-08 from Wolfgang Viechtbauer.

“The thing is, there are so many different elements to a forest plot (squares, lines, polygons, text, axes, axis labels, etc.), if I would add arguments to set the color of each element, things would really get out of hand (as far as I am concerned, there are already too many arguments to begin with) ...

*... what if somebody wants to have a different color for *one* of the squares and a different color for the other squares?”*

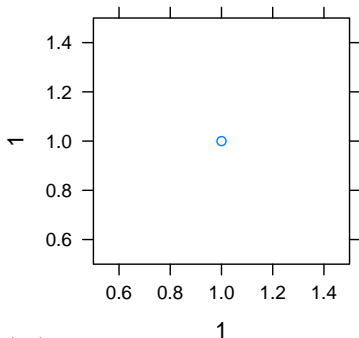
A low-level interface

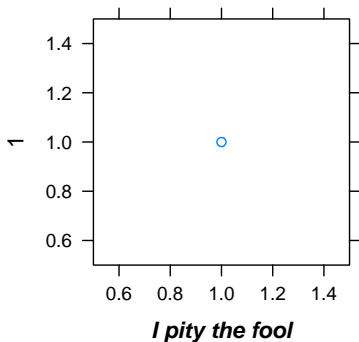
- It is convenient to have high-level interfaces for producing complex graphics, **but** ...
- It is **also** convenient to be able to treat the complex graphic as merely a collection of low-level components.
- The **grid** graphics system in R provides the infrastructure for such a low-level interface.
 - A **record** is kept of all components in a graphic.
 - Each component is **labelled**.
 - Components can be **accessed** by name and **modified**.

A low-level interface

```
> xyplot(1 ~ 1)
> grid.ls()

plot_01.background
plot_01.xlab
plot_01.ylab
plot_01.ticks.top.panel.1.1
plot_01.ticks.left.panel.1.1
plot_01.ticklabels.left.panel.1.1
plot_01.ticks.bottom.panel.1.1
plot_01.ticklabels.bottom.panel.1.1
plot_01.ticks.right.panel.1.1
plot_01.xyplot.points.panel.1.1
plot_01.border.panel.1.1
```





```
> grid.edit("plot_01.xlab",  
            label="I pity the fool",  
            gp=gpar(fontface="bold.italic"))
```

A low-level interface

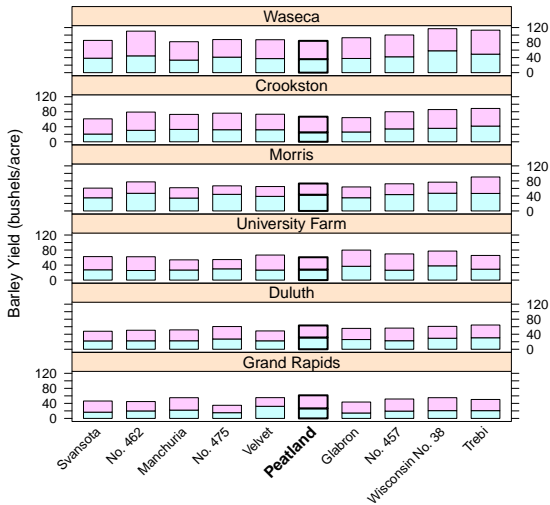
- This low-level interface allows ...
 - the user to **access** and **modify** all of the individual components of the plot.
 - the user to access and modify different **collections** of components.
 - the user to **record** and **programmatically** control a set of changes.

```
> barchart(yield ~ variety | site, data = barley,
           groups = year, layout = c(1,6), stack = TRUE,
           ylab = "Barley Yield (bushels/acre)",
           scales = list(x = list(rot = 45)))

> grid.edit("barchart.pos.6.rect",
           grep=TRUE, global=TRUE,
           gp=gpar(lwd=2))

> fontface <- rep("plain", 10)
> fontface[6] <- "bold"
> cex <- rep(.8, 6)
> cex[6] <- 1
> grid.edit("ticklabels.bottom.panel", grep=TRUE,
           gp=gpar(fontface=fontface, cex=cex))
```

Complex Statistical Graphics



Other important details

- For this sort of low-level approach to work, a couple of other things are necessary ...
 - The people who write high-level graphics functions have to name everything that they draw (or allow me svn access to their code).
 - **lattice** now has an explicit naming scheme.
 - The naming schemes have to be documented and/or functions provided to assist with generating component names
 - The **R-forge** page for **lattice** has a document describing its naming scheme and there is a `trellis.grobname()` function.
 - There need to be tools to help with identifying the components of a graphic.

grid debugging tools

- The `grid.ls()` function lists the components in a **grid** graphic.
- The `showGrob()` function draws labelled rectangles to show the location and names of components.

grid debugging tools



Other possibilities

- Having access to all components of a graphic by name provides a platform for more general **post-processing** of the graphic.
- The **gridSVG** package allows post-processing of a **grid** graphic to add dynamic and interactive features (in an SVG+javascript format).

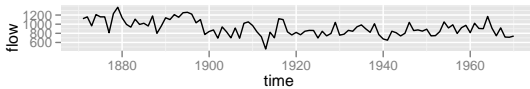
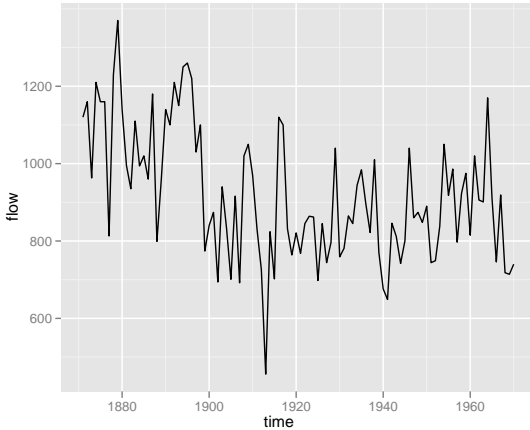
The `gridSVG` package

```
> library(ggplot2)
> library(gridSVG)

> df <- data.frame(time=as.numeric(time(Nile)),
                    flow=as.numeric(Nile))

> thePlot <- ggplot(df, aes(x=time, y=flow)) + geom_line()
> pushViewport(viewport(layout=grid.layout(2, 1,
                                           heights=c(4, 1))))
> pushViewport(viewport(layout.pos.row=1, name="topvp"))
> print(thePlot, newpage=FALSE)
> upViewport()
> pushViewport(viewport(layout.pos.row=2, name="bottomvp"))
> print(thePlot, newpage=FALSE)
> upViewport(2)
```

The `gridSVG` package



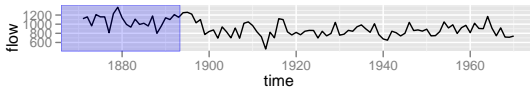
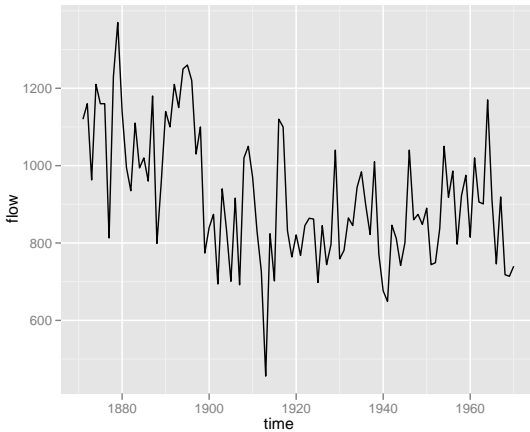
The `gridSVG` package

```
> downViewport("bottomvp")
> downViewport("panel-3-3")
> grid.rect(x=0, width=0.25, just="left",
            name="thumb",
            gp=gpar(col=rgb(0,0,1,.5), fill=rgb(0,0,1,.2)))

> grid.garnish("thumb",
              onmousedown="thumbDown(evt)")

> grid.script(filename="thumb.js")
```

The gridSVG package



```
> gridToSVG()
```

Interactive ggplot2 plot

A **grid** grob browser

- The **gridSVG** package can also be used to create tools that help with identifying plot components.

grid grob browser

Conclusions

- **grid** provides a low-level interface for accessing and modifying the individual components of a graphic by name.
- This low-level interface is a useful alternative to the typical high-level list-of-parameters for making arbitrary customisations to a graphic ...
- ... as long as the components have useful names and there are tools to discover/explore the names.
- A low-level interface is also useful for general post-processing of a graphic.
- A major benefit of providing access to named components of a graphic lies in what it lets **other people** do to the graphic.

Acknowledgements

- Velvet Ly helped to design the **lattice** naming scheme as part of her BScHons project.
- Simon Potter made many improvements to the **gridSVG** package as part of his BScHons project.
- The interactive **ggplot2** example makes extensive use of Duncan Temple Lang's **XML** package to post-process the SVG output produced by **gridSVG**.

References

- The **lattice** naming scheme is documented at <http://lattice.r-forge.r-project.org/Vignettes/src/naming-scheme/namingScheme.pdf>
- The **gridSVG** package is available from **R-forge** <https://r-forge.r-project.org/projects/gridsvg/>