Writing New Graphics Functions in **R** Using **grid**

Paul Murrell

The University of Auckland

Introduction

- **grid** provides a **low-level** graphics system.
- In the beginning, only **lattice** provided high-level plots based on **grid**, but some others are starting to appear—e.g., **hexbin** and **vcd**.
- **grid** provides support for producing high-level plots that are more flexible and reusable than the existing "traditional" plots.
- The aim of this talk is to make sure that **grid**'s features are fully exploited.
- The ultimate goal is to encourage the development of functions that produce **graphical components** that can be reused and recombined.

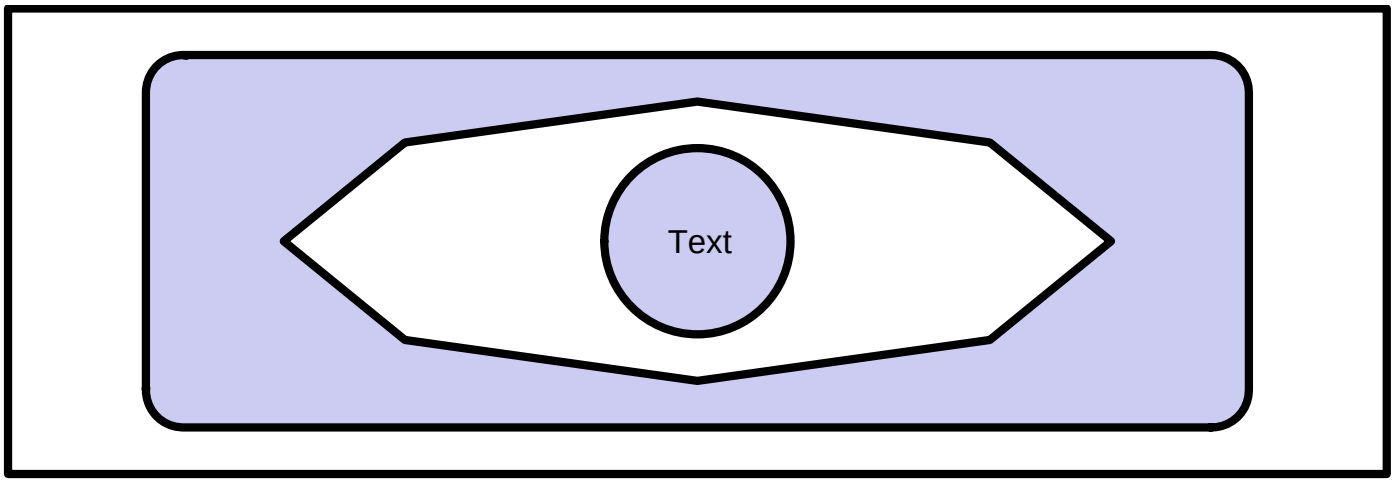
Overview

- A brief revision of important **grid** concepts.
  - Rseudo-code examples
- Writing graphics functions.
  - Support for embedding graphical components.
  - Support for annotating graphical components.
  - Support for customizing graphical components.
- Writing graphics classes.

grid Basics

- **grid** provides a standard set of **graphical primitives**.

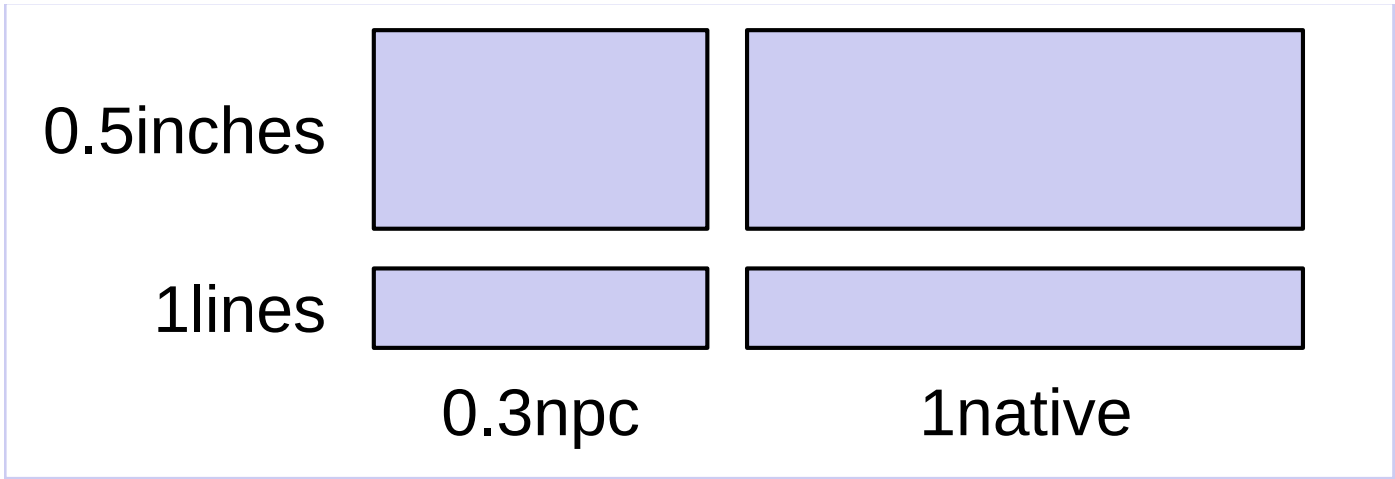
```
grid.rect(...)
grid.lines(...)
grid.polygon(...)
grid.circle(...)
grid.text(...)
```



grid Basics

- Graphical output can be positioned and sized relative to a large number of coordinate systems.

```
grid.rect(x=unit(0, "native"),
          y=unit(1.5, "lines"),
          height=unit(0.5, "inches"),
          ...)
```



grid Basics

- All drawing occurs in the current **grid viewport**.

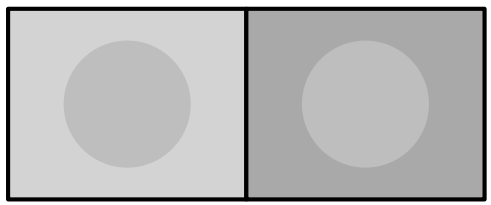
```
grid.text("centre")
pushViewport(viewport(...))
grid.text("centre")
popViewport()
```

grid Basics

- **grid** creates graphical objects (**grobs**) representing the graphical output.

```
grid.rect(..., name="background")
grid.circle(...)

grid.remove("background")
```

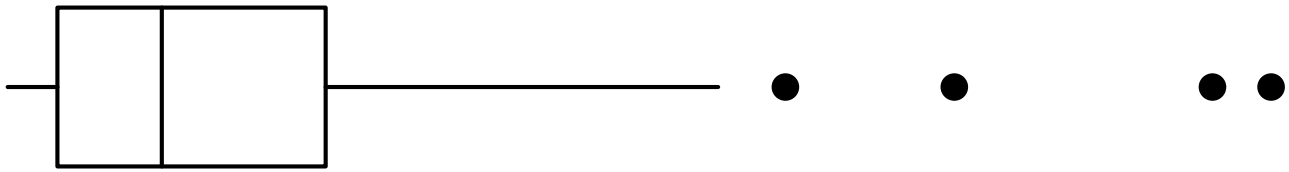


→

Writing graphics functions

- Consider a function that draws a boxplot.

```
x <- rexp(50)
gBoxplot(x)
```



Writing graphics functions

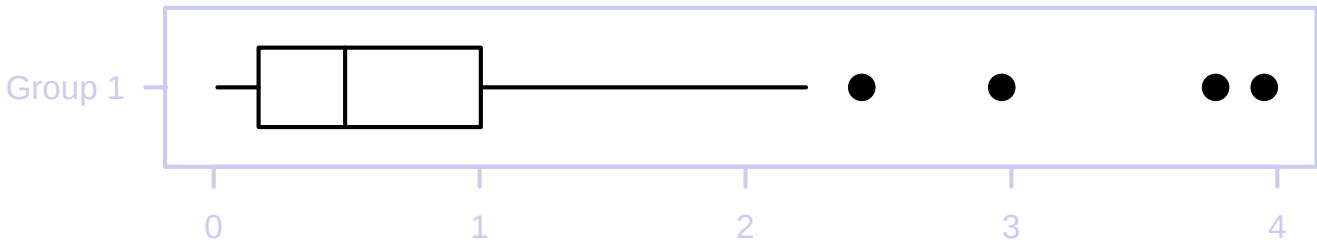
- A boxplot is just a rectangle, some lines, and possibly some points.

```
gboxplot <- function(...) {
  # inter-quartile range
  grid.rect(...)
  # median and whiskers
  grid.lines(...)
  grid.lines(...)
  # outliers
  if (outliers)
    grid.points(...)
}
```

Embedding

- Always draw in the current viewport.
  Do not assume you own the entire page.

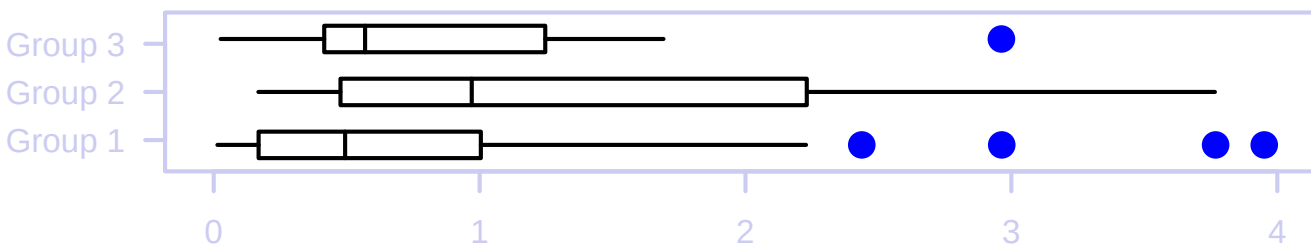
```
pushViewport(...)
gBoxplot(x, newpage=FALSE)
popViewport()
```



Embedding

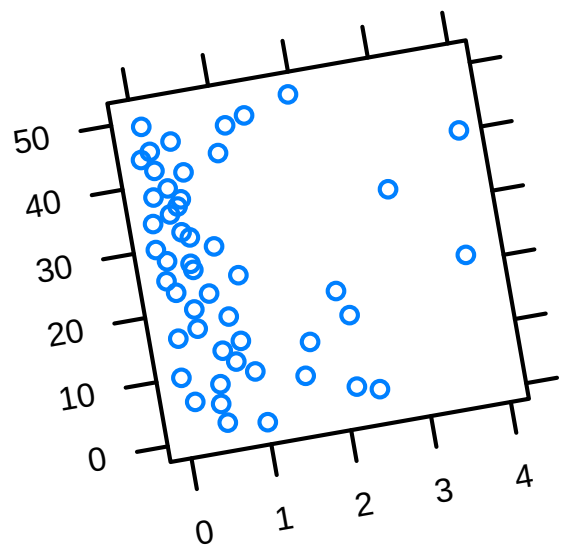
- Avoid absolute units. Will your output scale nicely? Only use absolute units if you have a good reason.

```
pushViewport(...)
gBoxplot(x1, ...)
gBoxplot(x2, ...)
gBoxplot(x3, ...)
popViewport()
```



A Lattice Example


```
lplot <- xyplot(...)
pushViewport(...)
print(lplot, newpage=FALSE)
popViewport()
```



Annotating

- Use `upViewport()` not `popViewport()`.
  Leave your viewports for others to use.
- Name your viewports.
  Make it easy for others to navigate within the coordinate systems that you create.

A "Bloated Boxplot" Example


```
gBloatedBoxplot <- function(...) {
  # inter-quartile range and median
  pushViewport(viewport(..., name="IQR"))
  grid.rect(...)
  grid.lines(...)
  upViewport()
  # median and whiskers
  grid.lines(...)
  # outliers
  if (outliers)
    grid.points(...)
}
```

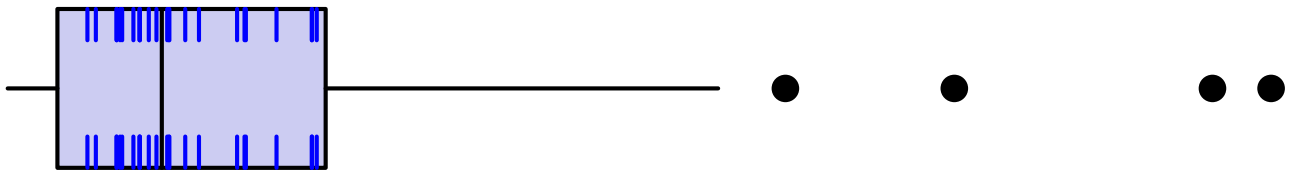
A "Bloated Boxplot" Example


```
gBloatedBoxplot(x)
current.vpTree()
```

```
viewport[ROOT]->(viewport[IQR])
```

A "Bloated Boxplot" Example


```
gBloatedBoxplot(x)
downViewport("IQR")
grid.segments(...)
upViewport()
```



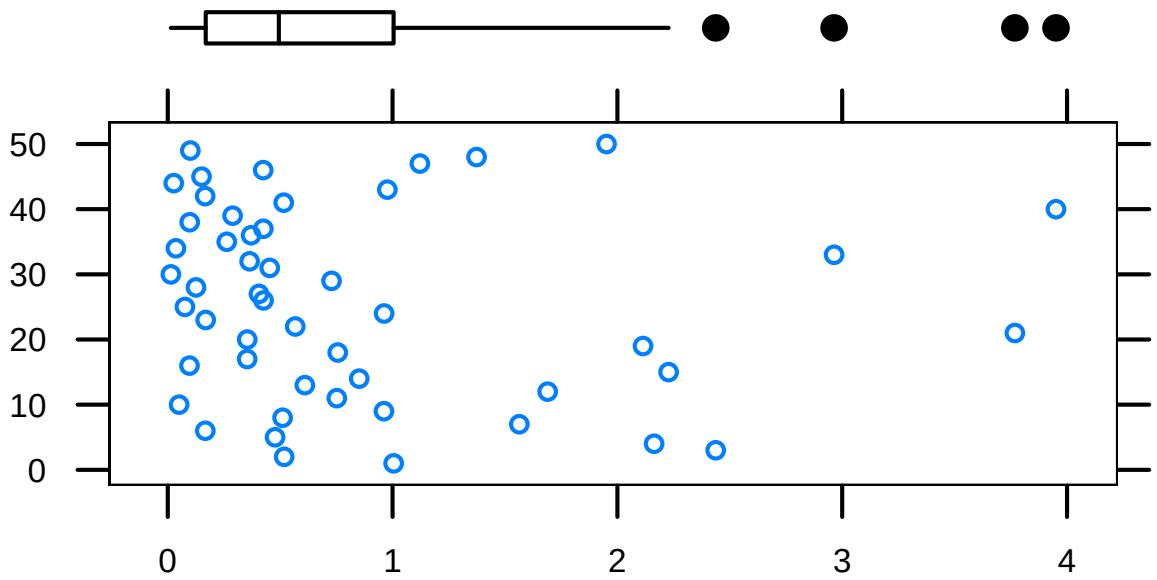
A Lattice Example


```
xyplot(x ~ 1:50)
current.vpTree()
```

```
viewport[ROOT]->(
  viewport[plot1.toplevel.vp]->(
    viewport[plot1.],
    viewport[plot1.panel.1.1.off.vp],
    viewport[plot1.panel.1.1.vp],
    viewport[plot1.strip.1.1.off.vp],
    viewport[plot1.xlab.vp],
    viewport[plot1.ylab.vp]))
```

A Lattice Example


```
xyplot(1:50 ~ x)
downViewport("plot1.panel.1.1.off.vp")
gBoxplot(x, ..., newpage=FALSE)
upViewport(0)
```



Customizing

- Name your graphical objects.
  Avoid having to provide arguments for every piece within your graphic.
  These are *some* of the arguments accepted by the traditional `bxp()` function:

  ```
  boxlty boxlwd boxcol boxfill
  medlty medlwd medpch medcex medcol medbg
  whisklty whisklwd whiskcol
  staplelty staplelwd staplecol
  outlty outlwd outpch outcex outcol outbg
  ```

  Make it easy/possible for others to access the objects that you create.

The Simple Example


```
gboxplot <- function(...) {
  # inter-quartile range
  grid.rect(..., name="IQR")
  # median and whiskers
  grid.lines(..., name="median")
  grid.lines(..., name="whiskers")
  # outliers
  if (outliers)
    grid.points(..., name="outliers")
}
```

Customizing


```
gBoxplot(x)
grid.edit("IQR", ...)
grid.edit("median", ...)
grid.edit("whiskers", ...)
grid.edit("outliers", ...)
```

A Lattice Example

These are *some* of the arguments accepted by the `xyplot()` function:

```
formula, data, panel,
allow.multiple, outer, aspect, as.table,
between, groups, key, auto.key, legend,
layout, main, page, par.strip.text,
prepanel, scales, skip, strip,
sub, xlab, xlim, ylab, ylim,
drop.unused.levels, par.settings,
perm.cond, index.cond, default.scales,
panel.groups, subscripts, subset
```

A Lattice Example


```
xyplot(x ~ 1:50)
getNames()
```

```
 [1] "GRID.GROB.1634" "plot1.xlab"
 [3] "plot1.ylab"     "GRID.GROB.1635"
 [5] "GRID.GROB.1636" "GRID.GROB.1637"
 [7] "GRID.GROB.1638" "GRID.GROB.1639"
 [9] "GRID.GROB.1640" "GRID.GROB.1641"
[11] "GRID.GROB.1642"
```

Hierarchical graphical objects (gTrees)

- Graphical objects can be combined to form hierarchies.

```
gTreeBoxplot <- function(...) {
  grid.draw(
    gTree(children=gList(rectGrob(...),
                         linesGrob(...),
                         linesGrob(...),
                         pointsGrob(...)),
          ...))
}
```

A hierarchical boxplot


```
gTreeBoxplot(x1, ..., name="boxplot")
getNames()
```

```
[1] "boxplot"
```

```
childNames(grid.get("boxplot"))
```

```
[1] "IQR"      "median"   "whiskers"
[4] "outliers"
```

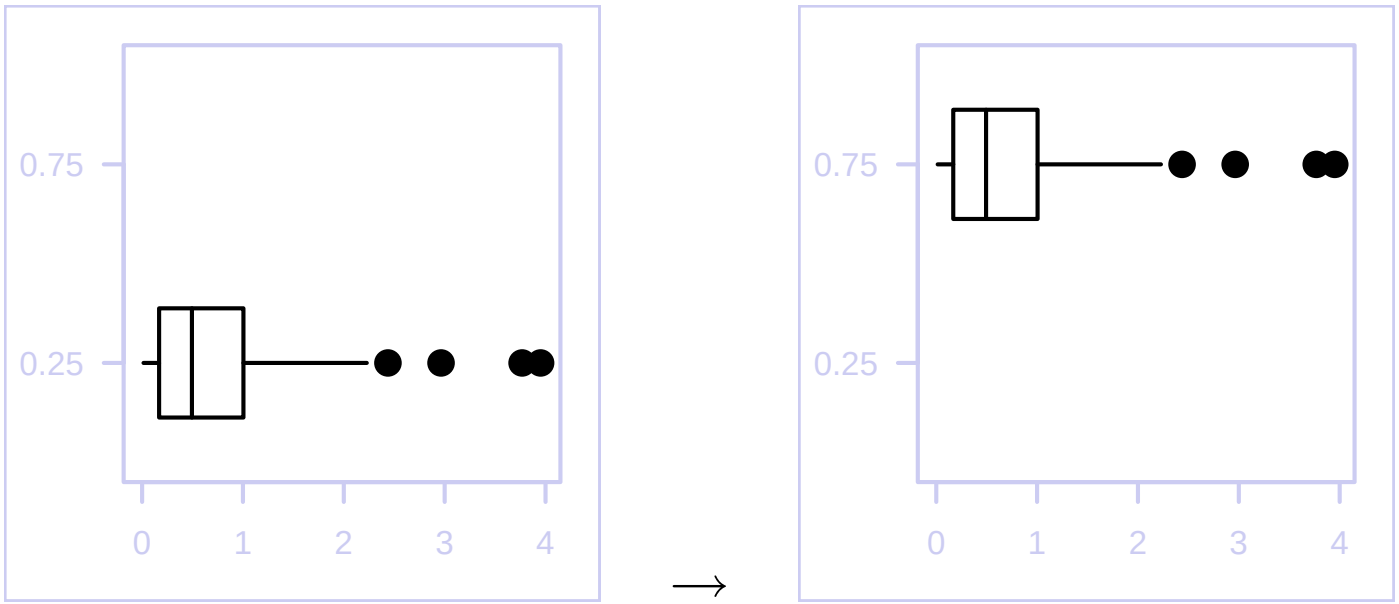
New Graphics Classes (and Methods)

- A graphical object can be given a class, which allows a high-level interface to be defined.

```
gTreeBoxplot <- function(x, at, ...) {
  grid.draw(
    gTree(children=gList(rectGrob(...),
                         linesGrob(...),
                         linesGrob(...),
                         pointsGrob(...)),
          x=x, at=at, ..., cl="gBoxplot"))
}
editDetails.gBoxplot <- function(...) {
  ...
}
```

A boxplot class


```
gClassBoxplot(x, at=0.25, ...,
              name="boxplot")
grid.edit("boxplot", at=0.75)
```



Summary

- New graphics functions are easy
- Make sure others can ...
  - ... **embed** your output
  - ... **annotate** your output
  - ... **customize** your output
- New graphics classes are possible