grid Graphics and Programming

Paul Murrell

The University of Auckland

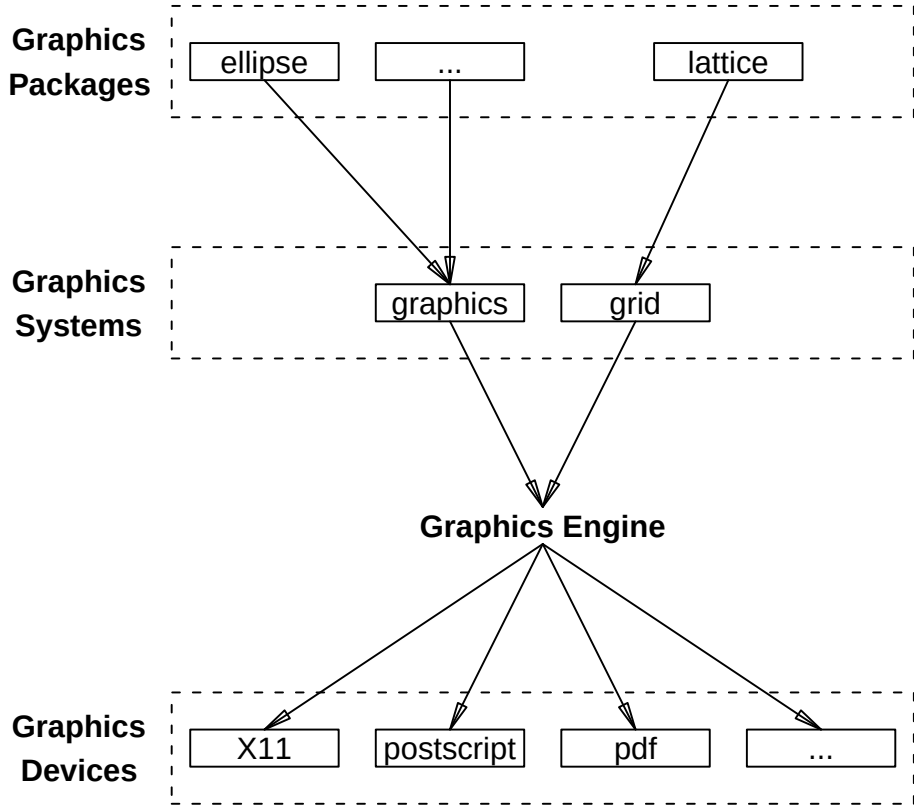
useR! 2004, Vienna

Talk Overview

- Introduction to grid
- Important grid concepts
- Sketching with grid
- Annotating with grid
- Editing with grid
- Combining grid with traditional graphics
- Developing new graphics with grid

Introduction to grid

The Structure of R Graphics



What is grid?

- grid is an alternative to the traditional graphics system provided by the graphics package.
- grid provides low-level graphics functions for producing statistical graphics (at least).

- lattice provides high-level graphics functions using grid

Why grid?

- grid began life purely as support for lattice
- The traditional system has some annoying constraints (e.g., text rotation in margins)
- The traditional system has some annoying inconsistencies (e.g., the meaning of `col`)
- As grid has developed, it has opened up opportunities to do some things that were not conceivable with the traditional system (e.g., interactive editing)

Uses for grid

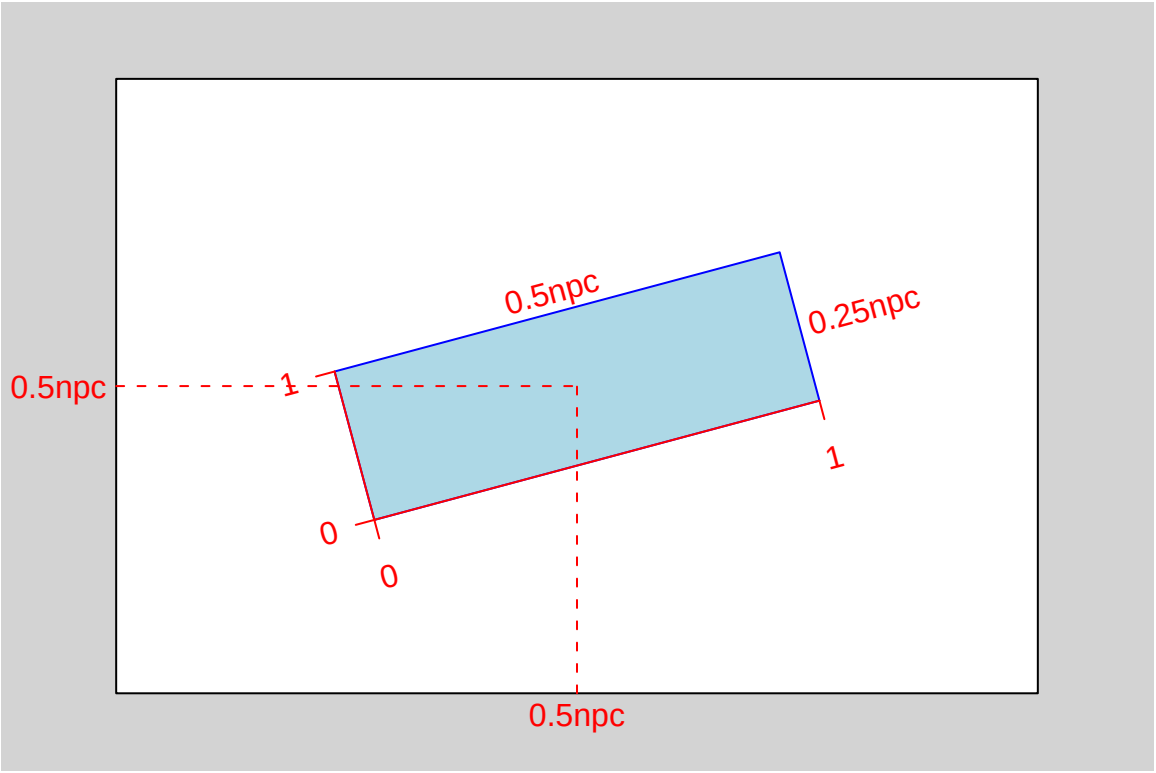
- As a drawing program
- Annotating lattice plots
- Editing lattice plots
- Tricky annotations of traditional plots (gridBase)
- Tricky arrangements of traditional plots (gridBase)
- Develop new graphics functions/components

Important grid Concepts

Viewports

A viewport is a rectangular region.

```
> viewport(x = 0.5, y = 0.5, width = 0.5, height = 0.25, angle = 15)
viewport[GRID.VP.1]
```



Pushing, Popping, Downing, and Upping

The `viewport()` function only creates a description of a viewport. The viewport description must be pushed in order to create a region on the device.

```
> pushViewport(viewport(x = 0.5, y = 0.5, width = 0.5, height = 0.25,
+     angle = 15))
viewport[GRID.VP.8]
```

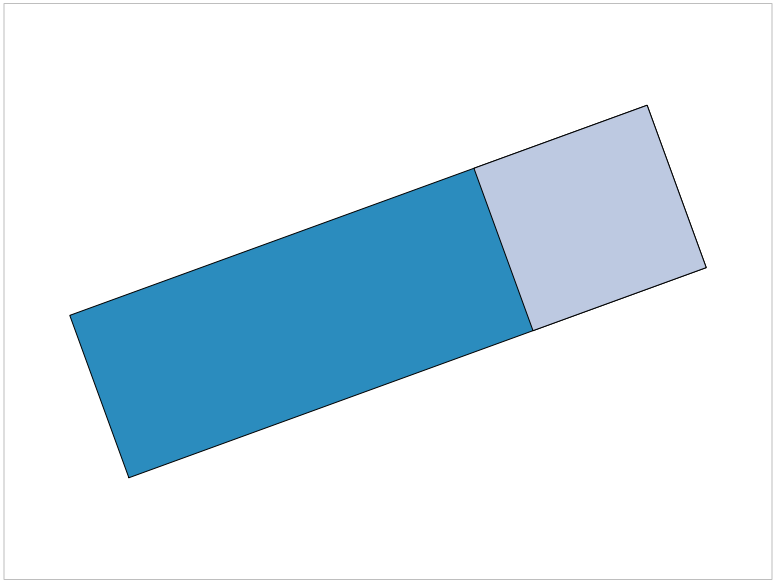
A viewport can be removed from a device by popping it.

```
> popViewport()
viewport[ROOT]
```

Pushing, Popping, Downing, and Upping

Viewports can be nested within each other.

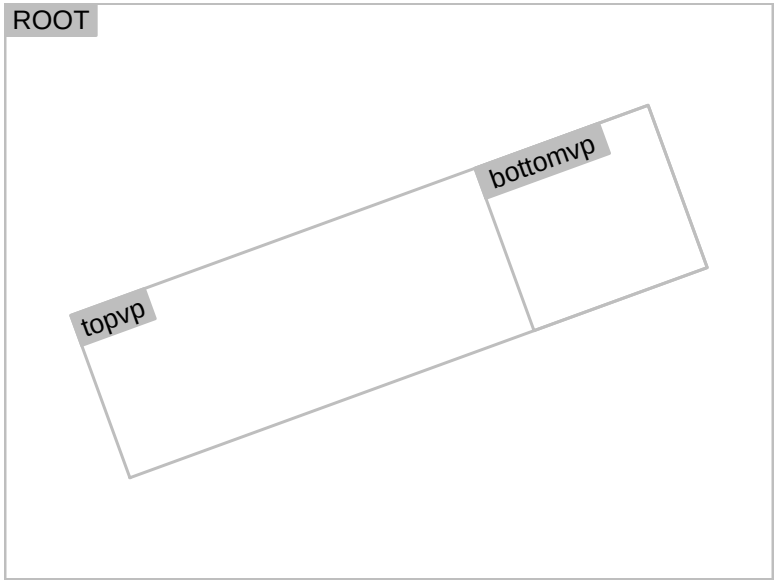
```
> pushViewport(viewport(width = 0.8, height = 0.3, angle = 20,
+     name = "topvp"))
viewport[topvp]
> grid.rect(gp = gpar(fill = rgb(43/255, 140/255, 190/255)))
> pushViewport(viewport(x = 1, width = 0.3, just = "right", name = "bottomvp"))
viewport[bottomvp]
> grid.rect(gp = gpar(fill = rgb(189/255, 201/255, 225/255)))
```



Pushing, Popping, Downing, and Upping

Instead of popping a viewport, it can be left in place, and we can navigate between viewports.

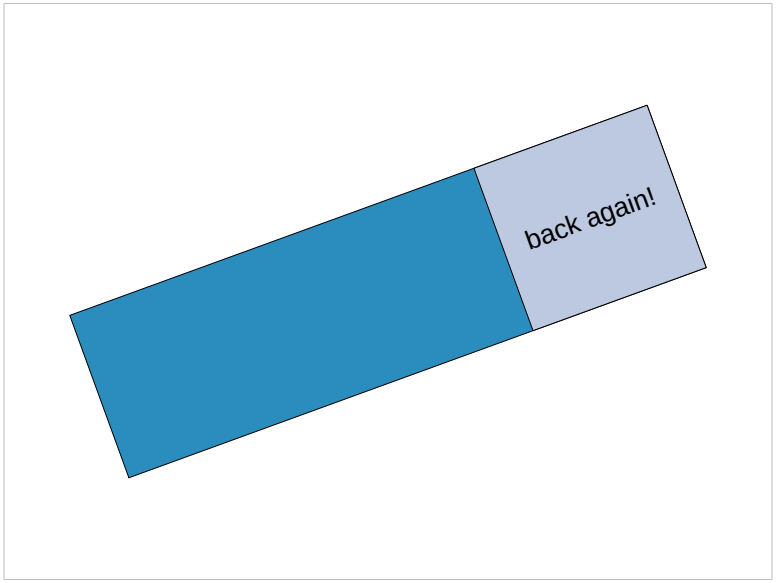
```
> upViewport(0)
viewport[ROOT]
> current.vpTree()
viewport[ROOT]->(viewport[topvp]->(viewport[bottomvp]))
```



Pushing, Popping, Downing, and Upping

Navigation amongst viewports makes use of viewport paths.

```
> downViewport(vpPath("topvp", "bottomvp"))
viewport[bottomvp]
> grid.text("back again!", gp = gpar(fontsize = 20))
```



Units and Coordinate Systems

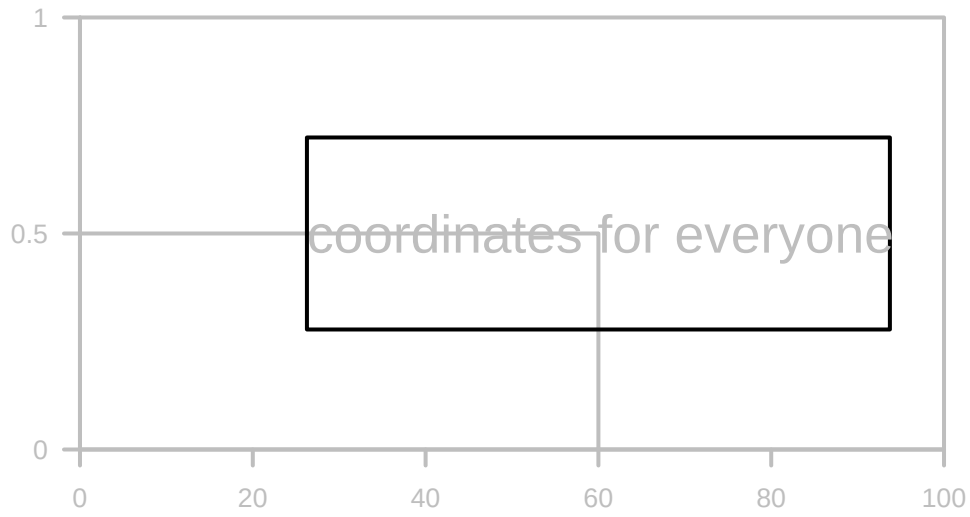
The `unit()` function associates values with coordinate systems.

```
> unit(1, "npc")
[1] 1npc

> unit(1:3/4, "npc")
[1] 0.25npc 0.5npc  0.75npc

> unit(1:3/4, "npc")[2]
[1] 0.5npc
> unit(1:3/4, "npc") + unit(1, "inches")
[1] 0.25npc+1inches 0.5npc+1inches  0.75npc+1inches
> min(unit(0.5, "npc"), unit(1, "inches"))
[1] min(0.5npc, 1inches)
```

Units and Coordinate Systems

Every viewport contains several coordinate systems.

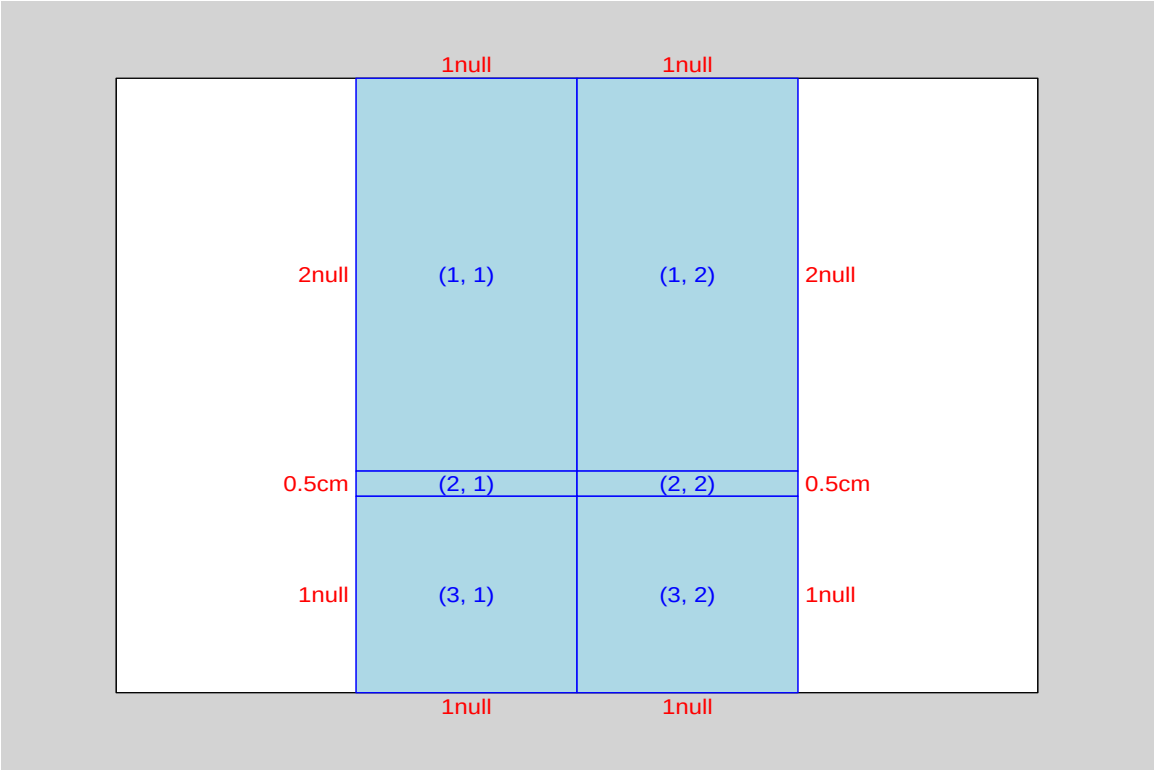
```
> pushViewport(viewport(xscale = c(0, 100)))
viewport[GRID.VP.9]
> pushViewport(viewport(x = unit(60, "native"), y = unit(0.5, "npc"),
+     width = stringWidth("coordinates for everyone"), height = unit(3,
+         "lines")))
viewport[GRID.VP.10]
```



Layouts

A layout divides a viewport into several rows and columns. You can specify different widths and heights of rows and columns.

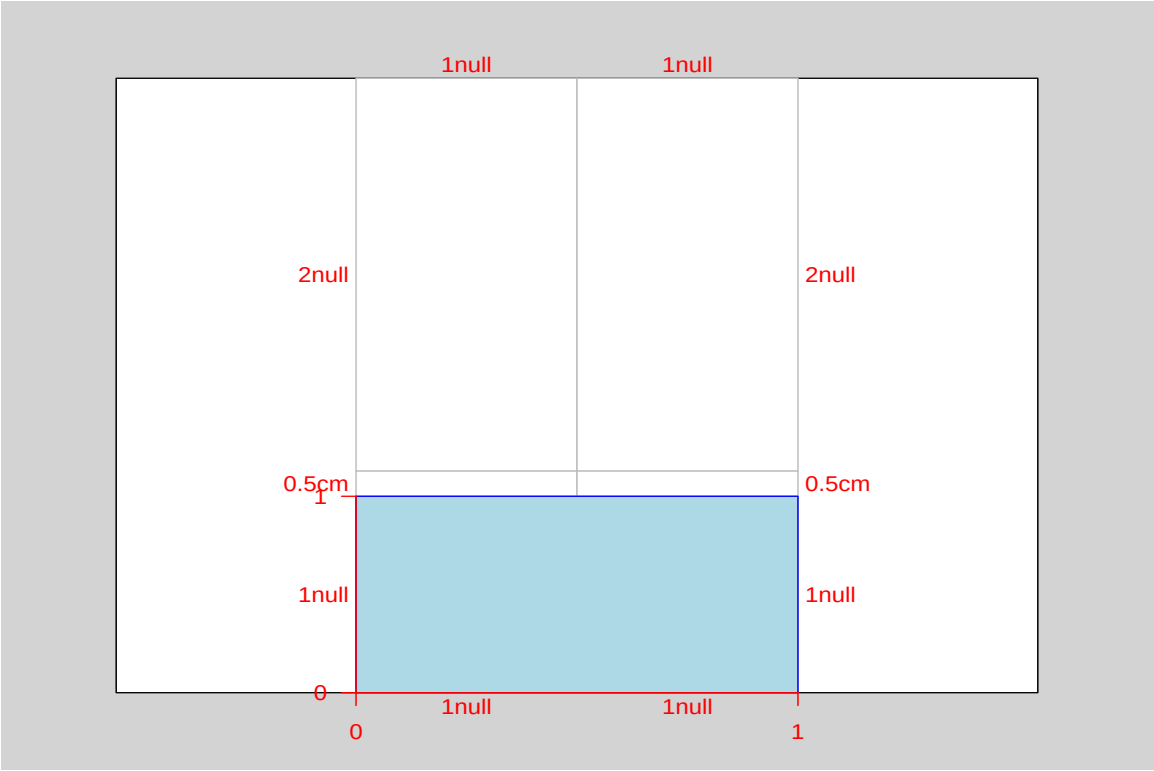
```
> grid.layout(3, 2, heights = unit(c(2, 0.5, 1), c("null", "cm",
+     "null")), respect = TRUE)
```



Layouts

Other viewports can occupy one or more cells of the layout.

```
> pushViewport(viewport(layout = grid.layout(3, 2, heights = unit(c(2,
+     0.5, 1), c("null", "cm", "null")), respect = TRUE)))
> pushViewport(viewport(layout.pos.row = 3))
```



grobs

A grob is a description of a something to draw.

```
> linesGrob(c(0.25, 0.25, 0.75), c(0.75, 0.25, 0.25))
[1] "lines[GRID.GROB.244]"
> rectGrob()
[1] "rect[GRID.GROB.245]"
> textGrob("A label")
[1] "text[GRID.GROB.246]"
```

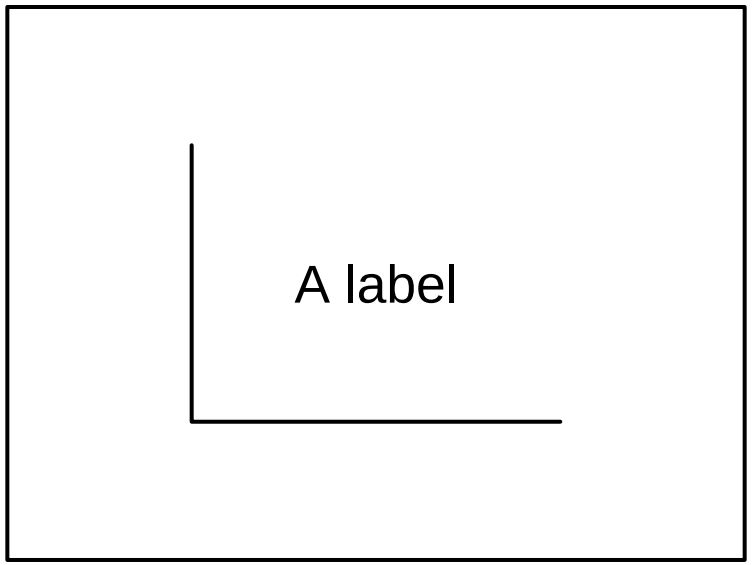
The `grid.draw()` function takes a grob and produces output on a device.

A gTree groups several grobs and allows them to be dealt with as a single object.

Drawing grobs

For every `*Grob()` function there is a `grid.*()` function which creates a grob and draws it.

```
> grid.lines(c(0.25, 0.25, 0.75), c(0.75, 0.25, 0.25))
> grid.rect()
> grid.text("A label")
```



grobs

The following grobs and gTrees are currently available.

`moveToGrob()`	`grid.move.to()`
`lineToGrob()`	`grid.line.to()`
`linesGrob()`	`grid.lines()`
`segmentsGrob()`	`grid.segments()`
`arrowsGrob()`	`grid.arrows()`
`polygonGrob()`	`grid.polygon()`
`circleGrob()`	`grid.circle()`
`rectGrob()`	`grid.rect()`
`textGrob()`	`grid.text()`
`pointsGrob()`	`grid.points()`
`xaxisGrob()`	`grid.xaxis()`
`yaxisGrob()`	`grid.yaxis()`

gpars

A gpar is a collection of graphical parameter settings.

```
> gpar(col = "red", lwd = 4, lty = "dashed")
$col
[1] "red"

$lwd
[1] 4

$lty
[1] "dashed"

attr(,"class")
[1] "gpar"
```

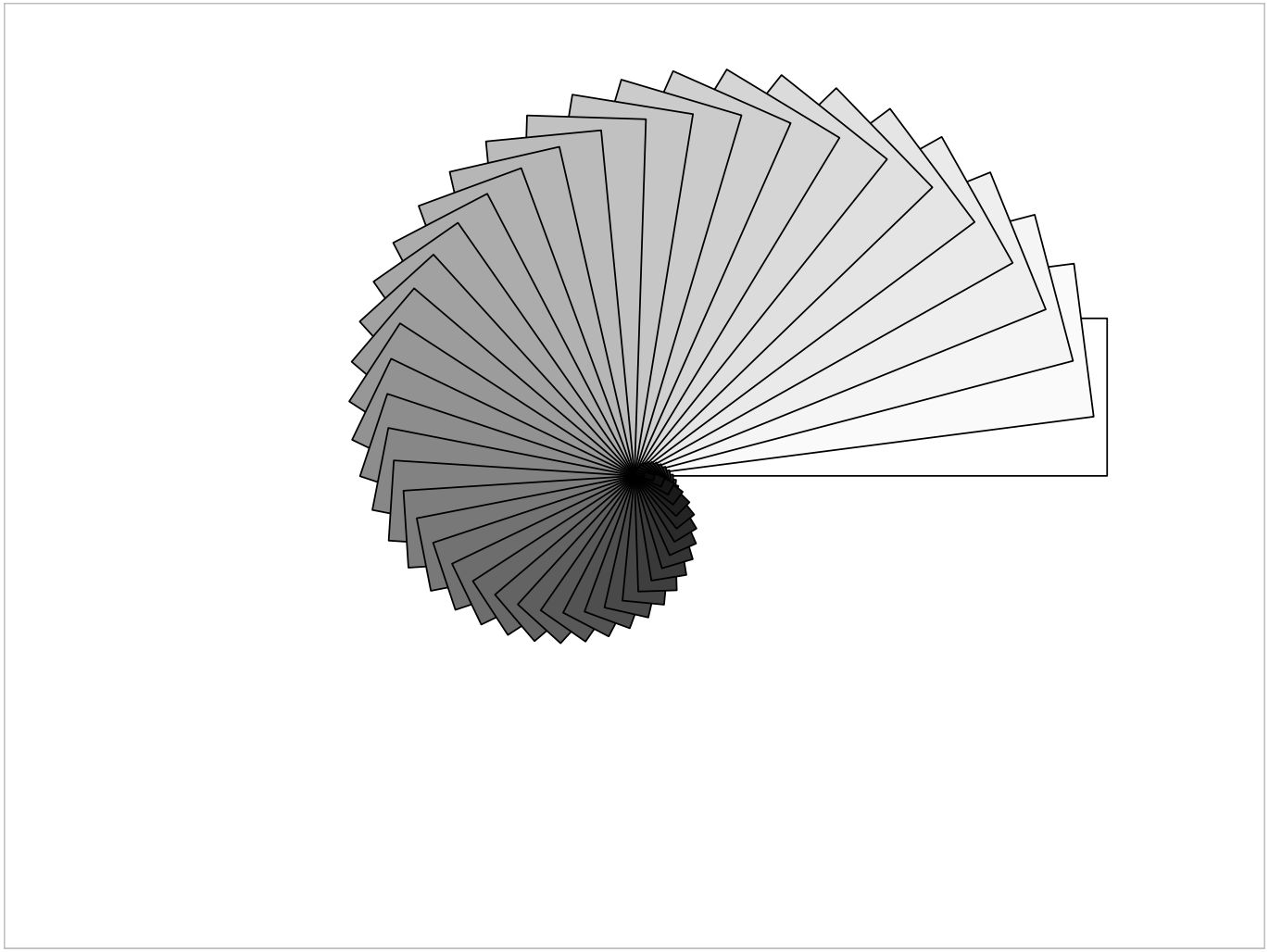
All viewports and grobs can have a gpar associated with them. The gpar settings in a viewport are inherited by grobs drawn in that viewport and by viewports pushed within the viewport.

Recap

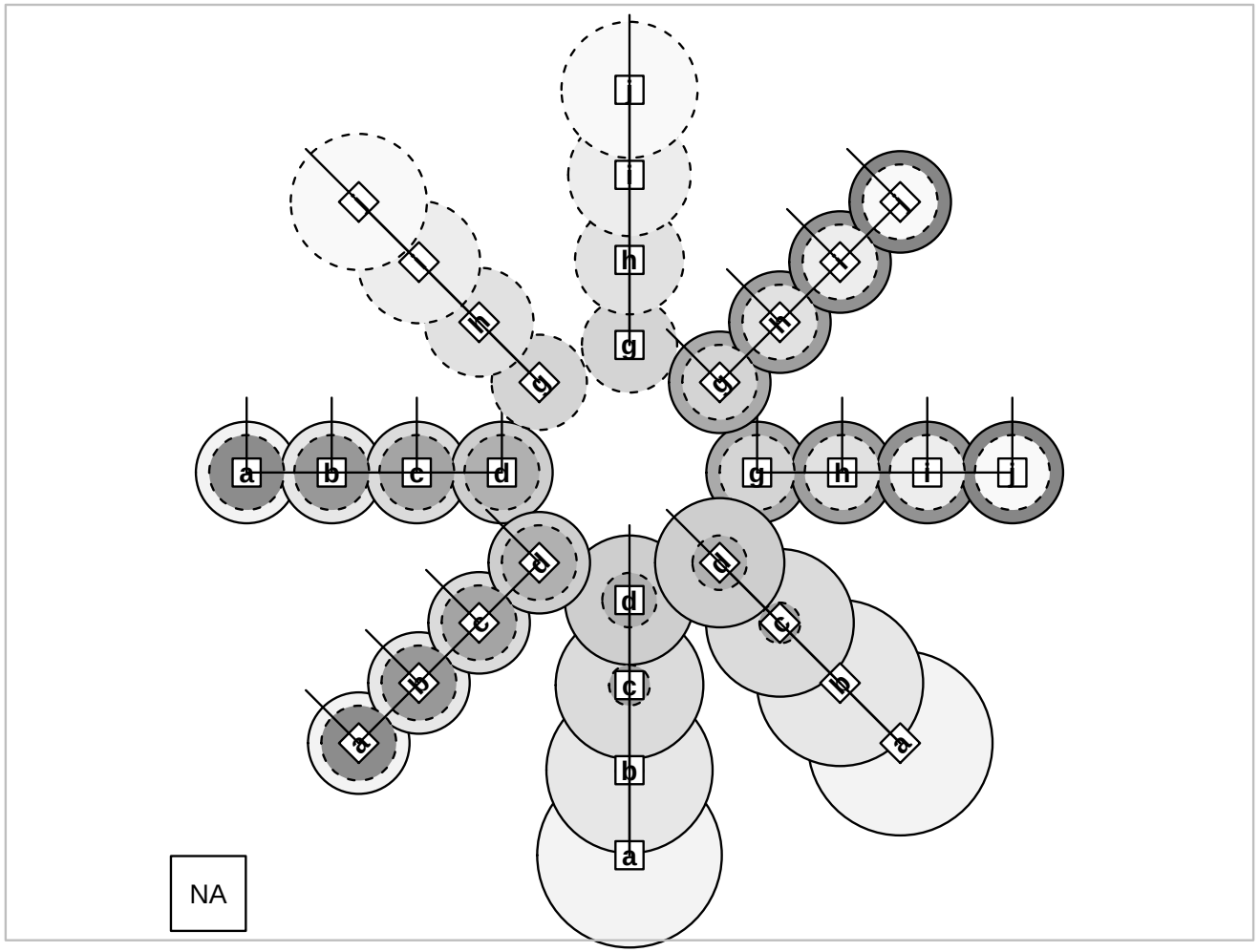
- viewports
- navigating the viewport tree
- units and coordinate systems
- layouts
- grobs
- gpars

Sketching with grid

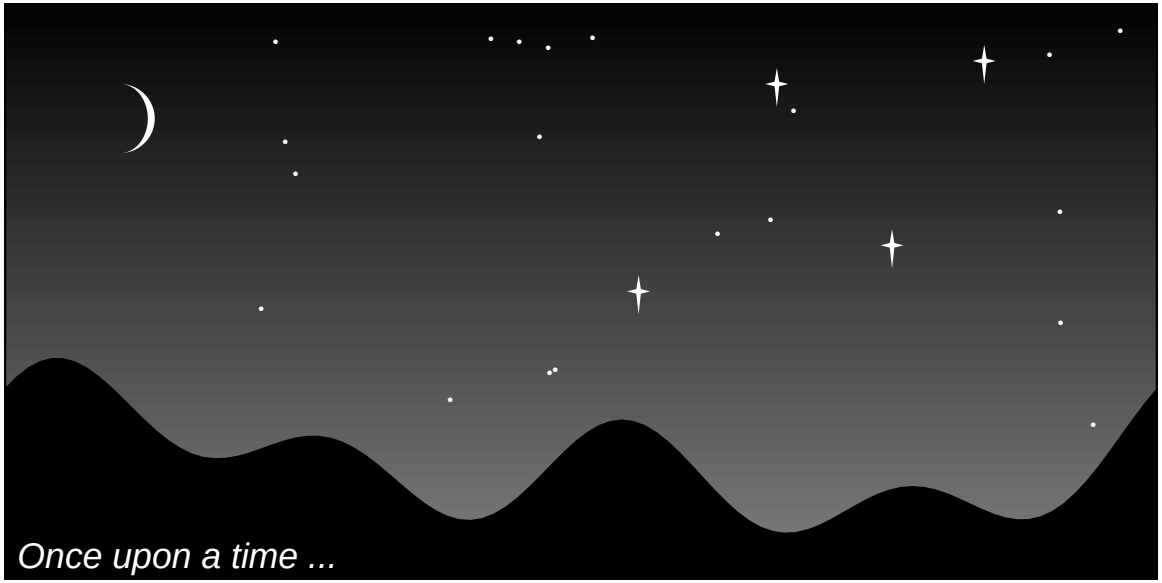
grid examples



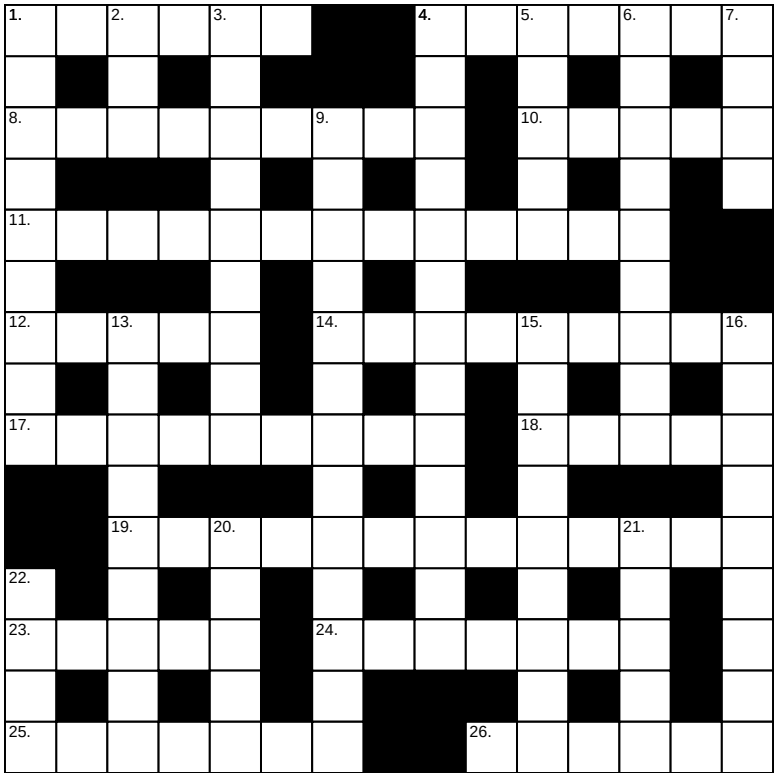
grid examples



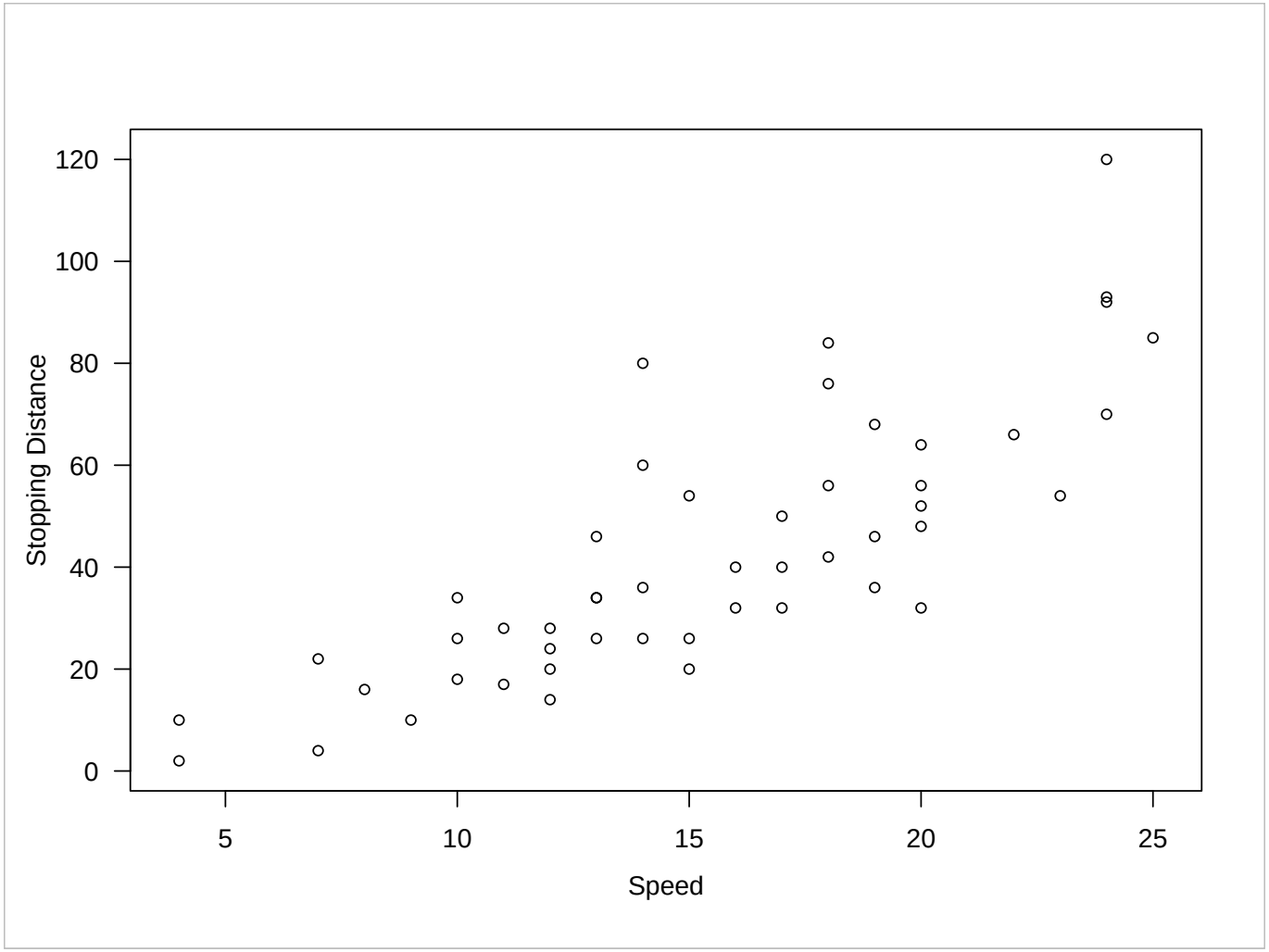
grid examples



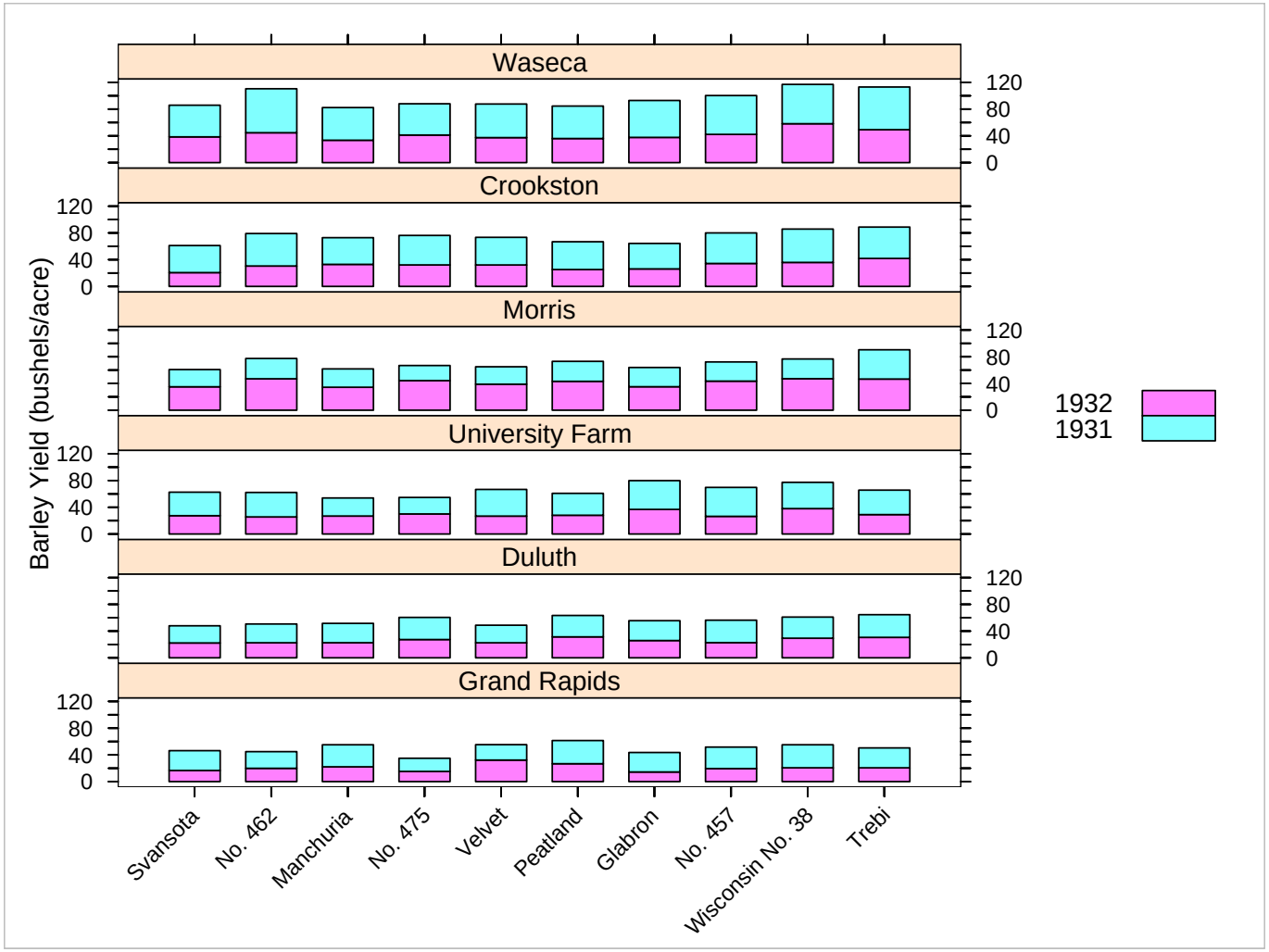
grid examples



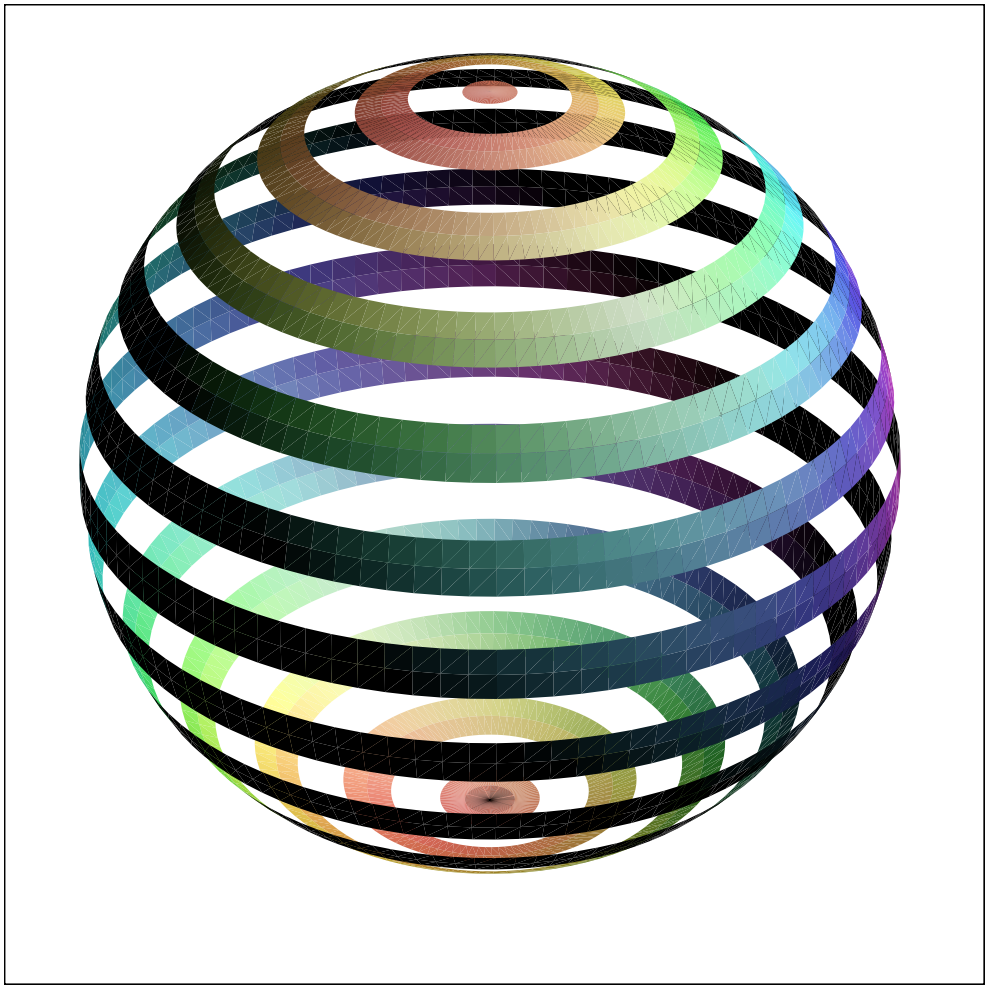
grid examples



lattice examples

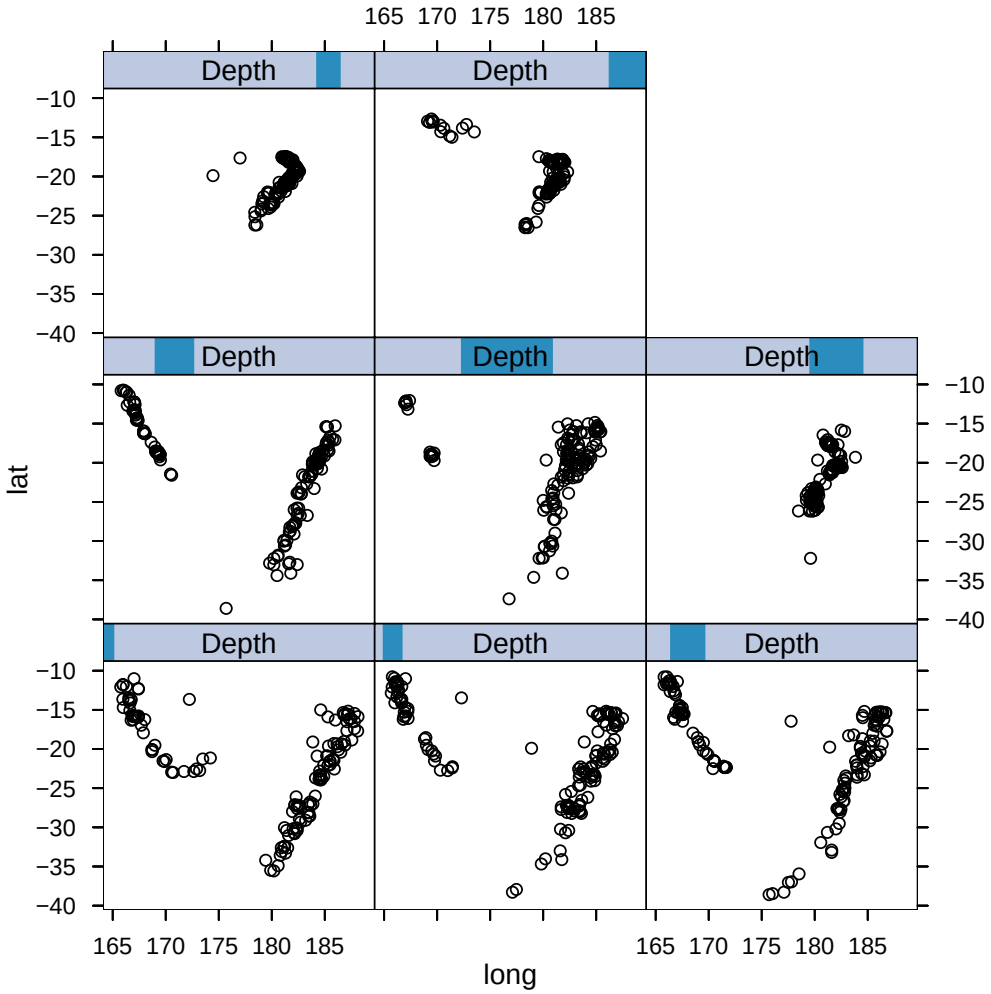


lattice examples



Annotating and Editing with grid

A lattice plot



Viewing the viewports

```
> current.vpTree()
```

```
viewport[ROOT]->(
  viewport[GRID.VP.502]->(
    viewport[GRID.VP.505]
    viewport[GRID.VP.507],
    viewport[GRID.VP.509],
    viewport[GRID.VP.510],
    viewport[GRID.VP.511],
    viewport[GRID.VP.512],
    viewport[GRID.VP.513],
    viewport[GRID.VP.515],
    ...
    viewport[panel.1],
    viewport[panel.2],
    viewport[panel.3],
    viewport[panel.4],
    viewport[panel.5],
    viewport[panel.6],
    viewport[panel.7],
    viewport[panel.8]))
```

Annotating the plot

Navigate to "panel 5":

```
> downViewport("panel.5")
```

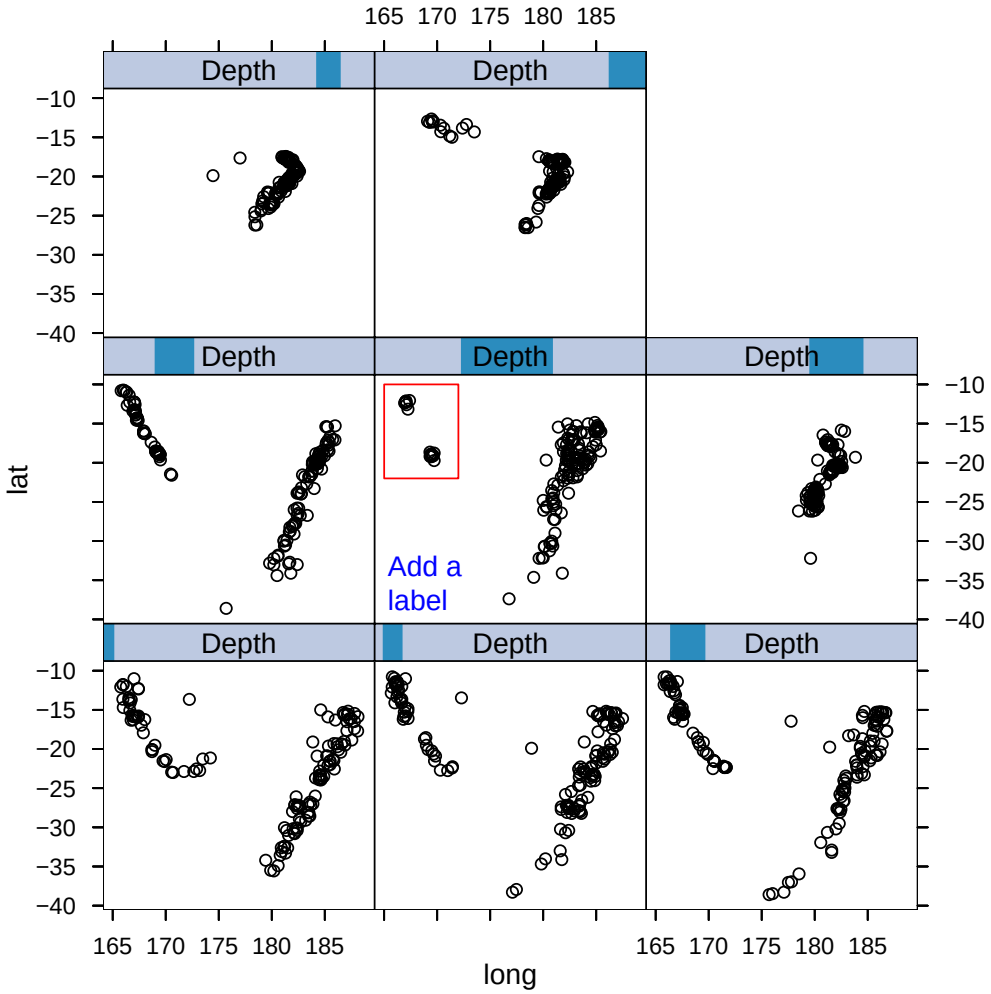
Add a text label in the bottom-left corner:

```
> grid.text("Add a\nlabel", x = unit(2, "mm"), y = unit(2, "mm"),
+     just = c("left", "bottom"), gp = gpar(col = "blue", lineheight = 1),
+     name = "ann.panel.5")
```

Draw a rectangle enclosing a data range:

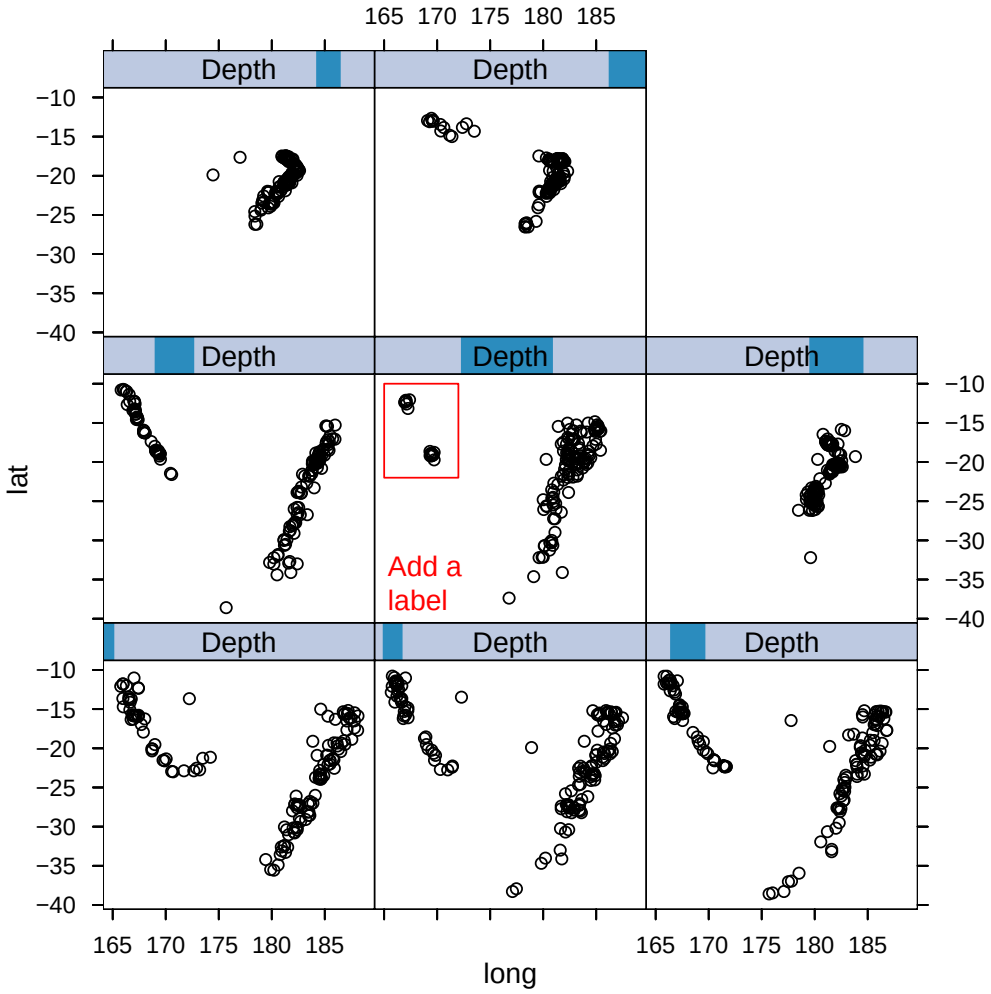
```
> grid.rect(165, -22, 7, 12, default = "native", gp = gpar(col = "red"),
+     just = c("left", "bottom"))
```

Annotating the plot



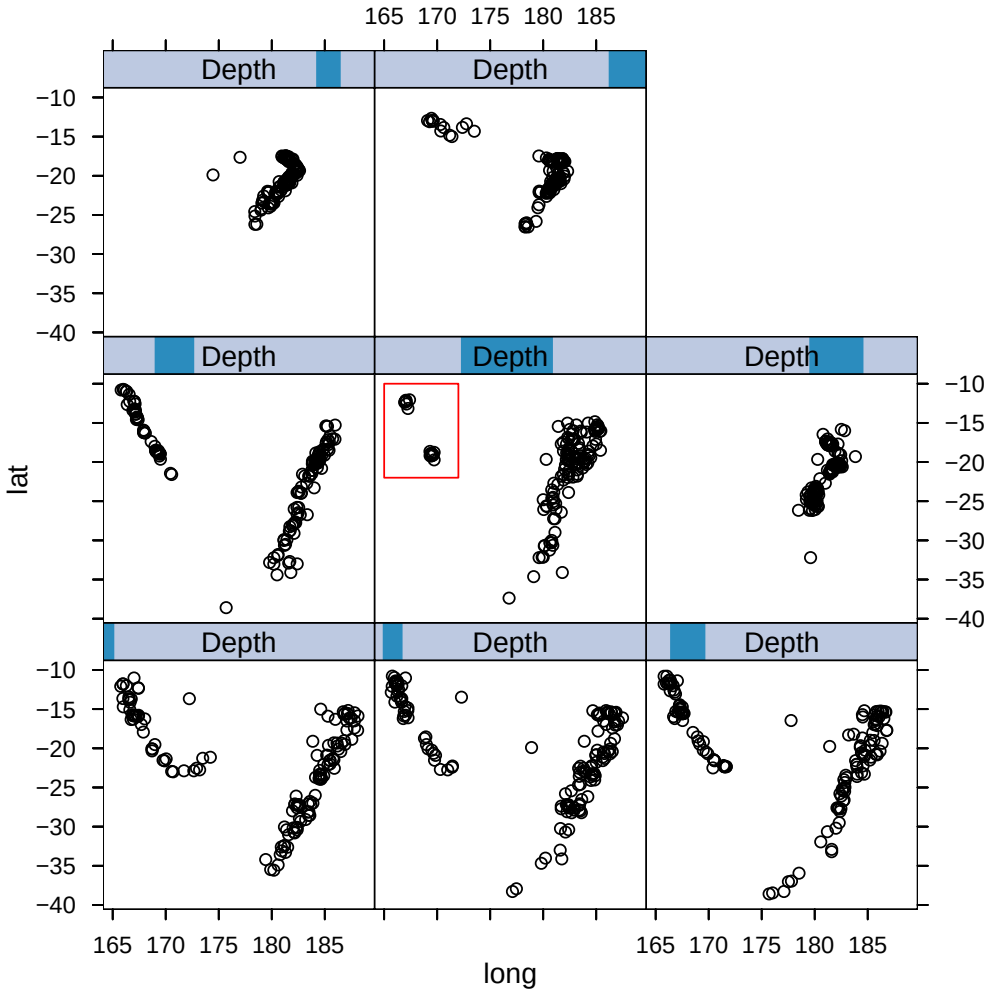
Editing the annotations

```
> grid.edit("ann.panel.5", gp = gpar(col = "red"))
```



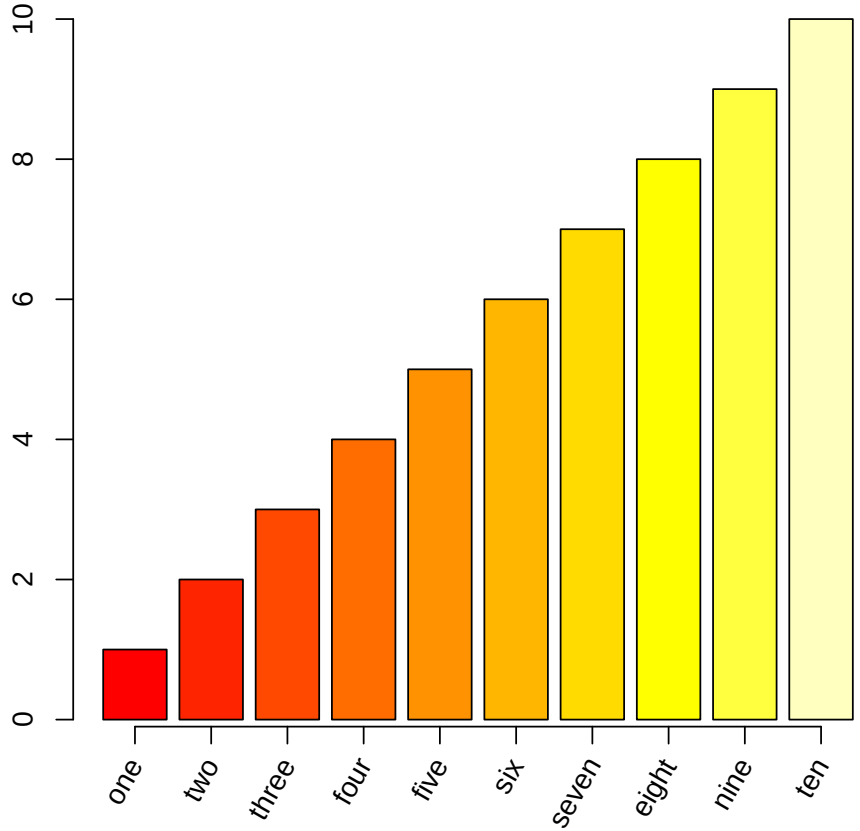
Editing the annotations

```
> grid.remove("ann.panel.5")
```

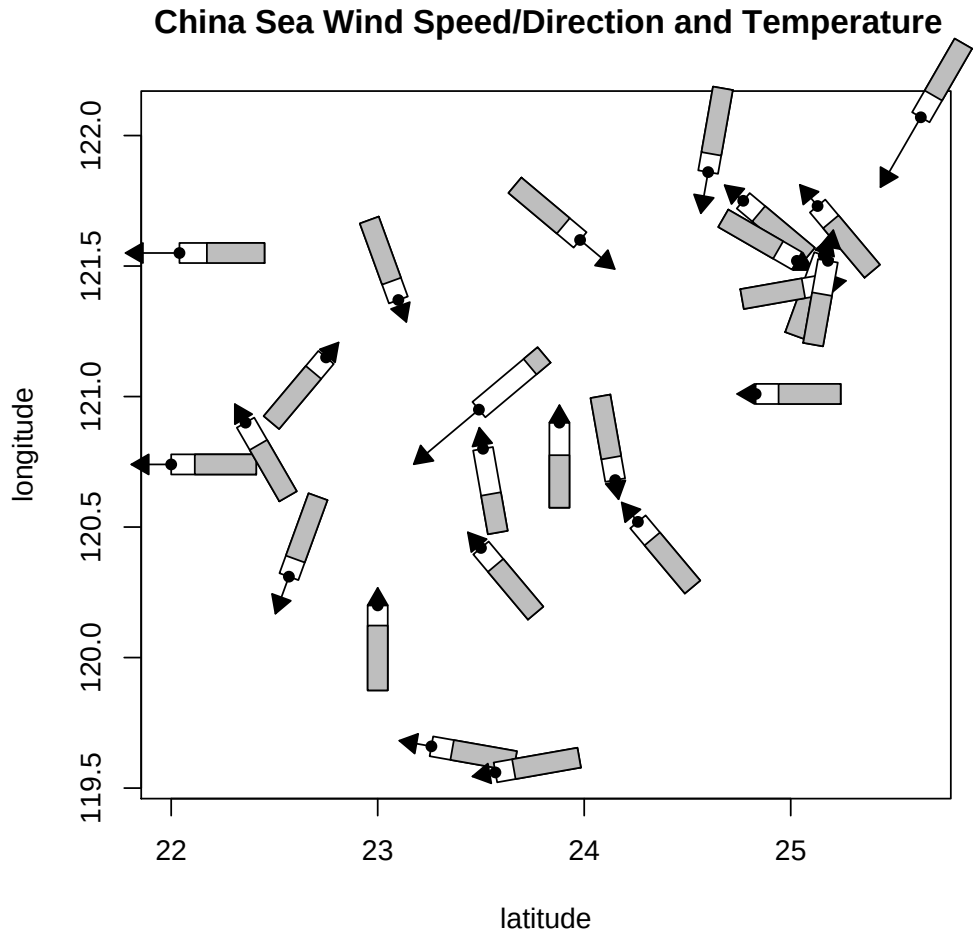


Combining grid and Traditional Graphics

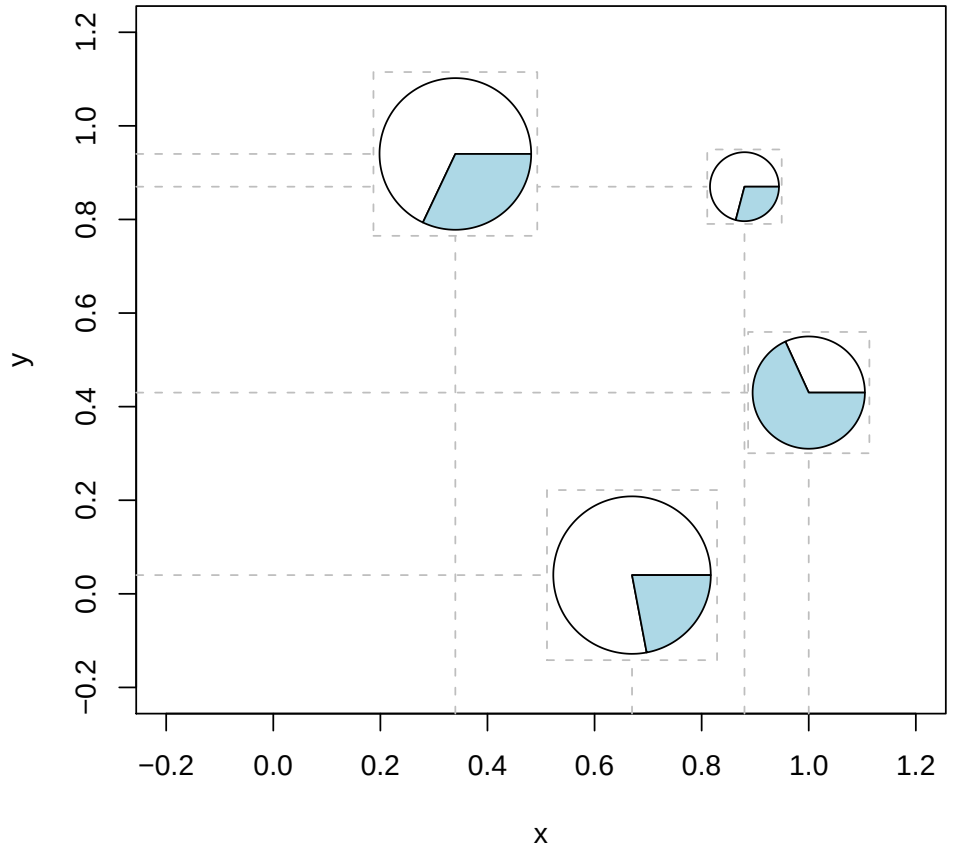
Annotating traditional plots with grid



Annotating traditional plots with grid



Arranging traditional plots with grid



Developing New Graphics with grid

Reasons for using grid

- Greater flexibility in specifying placement of graphical output and arrangements of plots (units and layouts)
- More consistency and generality (viewports and gpars)
- Better access to coordinate systems and output (navigating the viewport tree and interacting with grobs)
- Modular graphics; locations and sizes are declarative and the actual output depends on the viewport context. Graphical functions and grobs can be reused and embedded within other output.
- grobs provide a programatically editable persistent representation of graphical output; there is an API for working with graphical descriptions (`editGrob()`, `removeGrob()`, `addGrob()`, `getGrob()`, and `save()`).

Embedding lattice output