A new graphics API

Deepayan Sarkar

Indian Statistical Institute, Delhi

DSC 2014

Motivation

- Qt R Bindings (Michael Lawrence, $\approx$ 6 years)
- `qtpaint` — fast drawing API
- Wanted a high-level graphics system to go with it
- Eventually decided that needed something like `grid`
- Preferably something that's not too closely tied to Qt

Grid-like system

- Basic requirements
  - viewports
  - layouts
  - units
  - self-describing objects (widths/heights of strings)
- Doesn't need to be tied to a drawing system
- Implementation Based on abstract canvas (know pixel dimensions and DPI)

Grid-like system: tessella

```
> library(tessella)
> str(cont <- tcontext(x = 0, y = 0, w = 100, h = 100))
```

```
List of 5
 $ x       : num 0
 $ y       : num 0
 $ w       : num 100
 $ h       : num 100
 $ invert.y: logi FALSE
 - attr(*, "class")= chr "tcontext"
```

Grid-like system: `tessella`

```
> str(v <- tviewport(cont, x = 10, y = 10, w = 30, h = 40,
+                    xlim = c(0, 101), ylim = c(0, 1)))
```

```
List of 8
 $ parent : NULL
 $ context:List of 5
 $ x      : num 10
 $ y      : num 10
 $ w      : num 30
 $ h      : num 40
 $ xlim   : num [1:2] 0 101
 $ ylim   : num [1:2] 0 1
```

Grid-like system: tessella

```
> str(l <- tlayout(widths = c(5, -1, 5), heights = c(-1, 5),
+                  parent = v),
+     max.level = 1)
```

```
List of 6
 $ owidths       : num [1:3] 5 -1 5
 $ oheights      : num [1:2] -1 5
 $ widths        : NULL
 $ heights       : NULL
 $ respect.aspect: logi FALSE
 $ parent        :List of 8
 - attr(*, "class")= chr "tlayout"
```

Grid-like system: tessella

```
> str(refreshLayout(l), max.level = 1)
```

```
List of 6
 $ owidths        : num [1:3] 5 -1 5
 $ oheights       : num [1:2] -1 5
 $ widths         : num [1:3] 5 20 5
 $ heights        : num [1:2] 35 5
 $ respect.aspect: logi FALSE
 $ parent         :List of 8
 - attr(*, "class")= chr "tlayout"
```

Also `tgrob()` for objects with minimum dimensions (strings, legends)

Primitives

- Need to actually draw stuff at some point
- Primitives implemented by backends
- Sort of like graphics devices
- Uses environments
  (attached/detached for “dynamic namespace” behaviour)

Reference backend

```
> ls.str(graphics_primitives())
bbox_rot : function (w, h, rot)
opar :  NULL
tclip : function (vp)
tdpi :  num 72
tfinalize : function ()
tget_context : function ()
tinitialize : function (context, newpage = TRUE)
tlines : function (x, y, lty = 1, lwd = 1, col = 1, ..., vp)
tpoints : function (x, y, pch = 1, col = 1, fill = "transparent"
    lty = 1, ..., vp)
tpolygon : function (x, y, col = "black", fill = "transparent",
    fillOddEven = FALSE, ..., vp)
trect : function (xleft, ybottom, xright, ytop, fill = "transpar
    lty = 1, lwd = 1, ..., vp)
tsegments : function (x0, y0, x1 = x0, y1 = y0, lty = 1, lwd = 1
tstrheight : function (s, cex = 1, font = 1, family = "", rot =
tstrwidth : function (s, cex = 1, font = 1, family = "", rot = 0
```

Other backends

- qtbase - based on Qt's QGraphicScene/View API
- qtpaint - Michael's Qt-based fast drawing API
- ???

Potential advantages (over devices)

- Code once, render anywhere
- Possibility of more efficient implementations
- Make use of truly interactive backends

High-level package

- `yagpack`: Yet another graphics package
- Not unlike `lattice`
- Borrows ideas from `ggplot2`
  - "panel variables" are specified like aesthetics
  - "panel functions" are map + render layers
  - layers can be composed using +

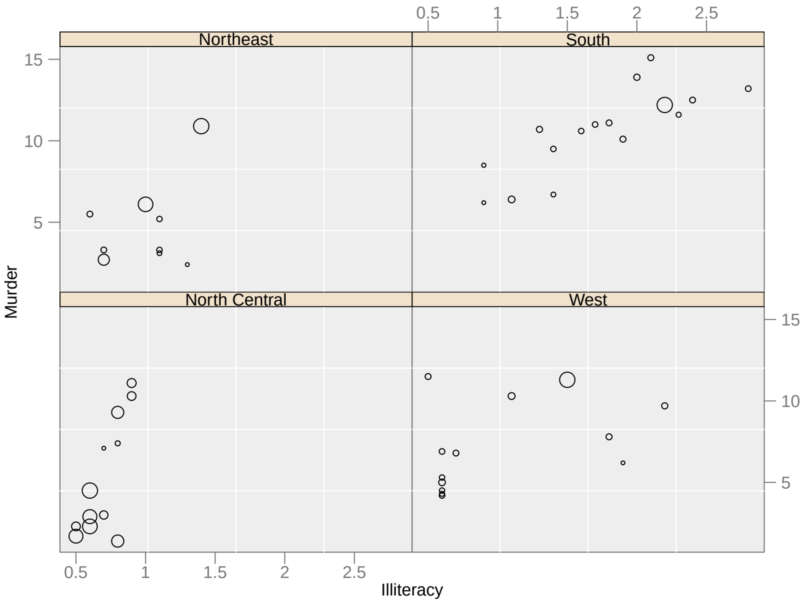
Example

```
> dstates <-
+     cbind(as.data.frame(state.x77),
+           Region = state.region,
+           State = I(rownames(state.x77)),
+           Area = state.area)
> str(dstates)
```

```
'data.frame': 50 obs. of  11 variables:
 $ Population: num  3615 365 2212 2110 21198 ...
 $ Income    : num  3624 6315 4530 3378 5114 ...
 $ Illiteracy: num  2.1 1.5 1.8 1.9 1.1 0.7 1.1 0.9 1.3 2 ...
 $ Life Exp  : num  69 69.3 70.5 70.7 71.7 ...
 $ Murder    : num  15.1 11.3 7.8 10.1 10.3 6.8 3.1 6.2 10.7 13.
 $ HS Grad   : num  41.3 66.7 58.1 39.9 62.6 63.9 56 54.6 52.6 4
 $ Frost     : num  20 152 15 65 20 166 139 103 11 60 ...
 $ Area      : num  50708 566432 113417 51945 156361 ...
 $ Region    : Factor w/ 4 levels "Northeast","South",..: 2 4 4
 $ State     :Class 'AsIs'  chr [1:50] "Alabama" "Alaska" "Arizo
 $ Area      : num  51609 589757 113909 53104 158693 ...
```

Example

```
> library(yagpack)
> p <-
+     yplot(data = dstates,
+           margin.vars = elist(Region), layout = c(2,2),
+           panel.vars = elist(x = Illiteracy,
+                              y = Murder,
+                              size = Area),
+           panel = ypanel.grid() + ypanel.xyplot(),
+           theme = yagp.theme("default"))
```



Northeast
South
North Central
West
Murder
Illiteracy
15
10
5
0.5
1
1.5
2
2.5

yagpack

- Work in progress, more or less functional now
- But why another system?
- Want to think about interaction.
    - I don't know how it should be done
    - Ideally some abstract API
    - I'll show some examples (graphicsEvent API, Qt)

Examples

- GraphicsEvent
  - Redraw/Animation: `graphics_redraw.R`
  - Layers: `graphics_layers.R`
- Qt backends
  - qtbase + qtpaint: `quilt.R`
  - Pure qtpaint: `qtpaint.R`

Summary

- Standard R graphics - graphicsEvent API
- What I would like
  - More device support
  - Mouse wheel events
  - Layers (two devices plotting on same surface)
- May give basic interactivity to vanilla R
- Qt probably better prototype for the long term

Summary

- Long-term goals
  - Code once, plot anywhere
  - Publication-quality static plots
  - Develop `yagpack` with support for linking etc.
  - Work on abstract interaction API ...
  - Similar Javascript canvas API, maybe generated by R?
  - ???

Development code

- `github.com/deepayan/tessella`
- `github.com/deepayan/yagpack`
- `github.com/ggobi/qtbase`
- `github.com/ggobi/qtpaint`
- `github.com/ggobi/qtutils`