Display Lists in R

Paul Murrell and Ross Ihaka

The University of Auckland

February 16 2007

What is a Display List?

A display list is a record of the **graphics commands** used to create a drawing.

- It is **not** a **backing store**.

```
windows(bg="cornsilk")
dev.control("inhibit")
plot(1:10)
# Cover/uncover window
# Resize window
```

- A graphics command may imply a considerable amount of calculation before drawing occurs, e.g., **coordinate system** transformations.

```
dev.control("enable")
plot(1:10)
# Resize window
```

What is a Display List **for**?

A display list is required when the drawing needs to be **automatically** redrawn.

- To explore **aspect ratios**.

```
# Resize device
```

- To copy between devices with different **sizes** or different **formats** (raster versus vector).

```
dev.print()
```

The Display List in R

The graphics commands recorded on the display list in R are **C-level** entry points for graphics functions.

```
> plot(1:10)
> myplot <- recordPlot()
> lapply(myplot[[1]], "[[", 1)
```

```
.Primitive("plot.new")
.Primitive("plot.window")
.Primitive("plot.xy")
.Primitive("axis")
.Primitive("axis")
.Primitive("box")
.Primitive("title")
```

All calculations in **C code** are rerun when the display list is rendered.

The **Problem** with the Display List in R

The display list in R is not always up to the job.

```
plot(1:10)
legend(1, 10, "An example", pch=1, bg="tan")
# Resize window
```

The problem is that calculations in **R code** are not recorded on the display list.

```
> lapply(recordPlot()[[1]], "[[", 1)
```

```
.Primitive("plot.new")
.Primitive("plot.window")
.Primitive("plot.xy")
.Primitive("axis")
.Primitive("axis")
.Primitive("box")
.Primitive("title")
.Primitive("rect")
.Primitive("plot.xy")
.Primitive("text")
```

The **Problem** with the Display List in R

The R code consists of ...

1. draw plot
2. calculate size and position of legend
3. draw legend

... but the display list only records ...

1. draw plot
2. draw legend

Another way to state the problem is that the **system** is deciding what to record on the display list and it is not smart enough.

The **Solution**

The `recordGraphics()` function records an arbitrary expression on the display list, along with an environment within which to evaluate the expression.

```
recordGraphics(expr, list, env)

expr: object of mode 'expression' or 'call' or
      an "unevaluated expression".
list: a list defining the environment in which
      'expr' is to be evaluated.
 env: An 'environment' specifying where R looks
      for objects not found in 'envir'.
```

In other words, let the **user** decide what goes on the display list.

The Solution in Action

Now calculations in R code can be captured on the display list.

```
plot(1:10)
recordGraphics(legend(1, 10, "An example",
                      pch=1, bg="tan"),
               list(), globalenv())
# Resize window
```

```
> lapply(recordPlot()[[1]], "[[", 1)
```

```
.Primitive("plot.new")
.Primitive("plot.window")
.Primitive("plot.xy")
.Primitive("axis")
.Primitive("axis")
.Primitive("box")
.Primitive("title")
.Primitive("recordGraphics")
```

The Solution in Action

Now calculations in R code can be captured on the display list.

```
> recordPlot()[[1]][[8]]
```

```
[[1]]
.Primitive("recordGraphics")

[[2]]
[[2]][[1]]
legend(1, 10, "An example", pch = 1)

[[2]][[2]]
list()

[[2]][[3]]
<environment: R_GlobalEnv>
```

The Solution in Action

It is possible to make, for example, the arrangement of plots much smarter.

```
windows(bg="cornsilk")
source("dynamic.R")
grid.dynamic()
# Resize window
```

The Solution in Action

A little more expertise is required to operate this mechanism correctly and/or efficiently.

Some approaches are more efficient than others.

```
plot(1:10)
recordGraphics(legend(1, 10, "An example", pch=1),
               list(), getNamespace("graphics"))
# Resize window
```

It is not hard to get it wrong.

```
plot(1:10)
recordGraphics(legend(1, 10, "An example", pch=1),
               list(), emptyenv())
```

The **Problem** with the Solution

The system is smart enough to avoid obvious problems ...

```
recordGraphics(recordGraphics(plot(1:10),
                              list(),
                              getNamespace("graphics")),
               list(), getNamespace("graphics"))
# Resize window
recordPlot()[[1]]
```

... but it is not a good idea to put actions with global side-effects on the display list.

```
recordGraphics({ plot(1:10); windows() },
               list(), getNamespace("graphics"))
# Resize window
```

Conclusions

- The original display list in R gave the **system** too much control.
- The `recordGraphics()` function gives the **user** too much power.
- There is **NOT** a happy middle ground.

The **other** Display List in R

the answer to Martin's question

The grid graphics system also maintains a display list. Whenever a grid function, e.g., `grid.circle()`, is called, the following happens:

1. A graphical object (grob) is created.
2. A call to the `drawGrob()` function is recorded on R's display list.
3. The grob itself is stored on grid's display list.

```
grid.circle()
recordPlot()[[1]]
getNames()
```

The **other** Display List in R

The `drawGrob()` function calls an appropriate `drawDetails()` method for the grob, so any code in a `drawDetails()` method will be rerun when R's display list is rendered.

```
grid.dynamic
dynamicGrob
body(drawDetails.dynamic)
```

Do **NOT** put a grid function inside a `recordGraphics()` call.

```
recordGraphics(grid.circle(), list(), globalenv())
# Resize window
recordPlot()[[1]]
getNames()
```

What is the **grid** Display List for?

R's display list is only designed for rerunning code.

Grid's display list is designed for several things:

- Modifying the drawing (including removing elements).
- Querying the drawing, e.g., determine where drawing has occurred.