



*DSC 2001 Proceedings of the 2nd International
Workshop on Distributed Statistical Computing
March 15-17, Vienna, Austria
<http://www.ci.tuwien.ac.at/Conferences/DSC-2001>
K. Hornik & F. Leisch (eds.) ISSN 1609-395X*

R Graphics

The Good, The Bad, and the Ugly

Ross Ihaka *

Abstract

The present R graphics system was originally written as a placeholder, which could be replaced when a better alternative became available. This temporary solution has now become so entrenched that it is probably no longer possible to remove it. In this paper we will look at some of what is possible in the R graphics system, how it could be improved, and what might not be fixable. We will also look at how the existing system might be augmented and see what tradeoffs might be involved in adding an additional graphics system.

1 The R Graphics System

Like much of the rest of R, the graphics system was written quickly, with minimal programming resources. The strategy at the time was to implement a basic graphics system and then to revisit the design of the graphics system at a later time. However, other R development tasks have consumed any resources which might have been available for this task, and no rewrite has taken place.

Because of the need to work quickly, it was decided to implement the existing API of the S graphics system. As a result, R now has a graphics system is largely compatible with that of the S system, as described in Becker and Chambers [1] or Becker, Chambers, and Wilks [3].

The S graphics system is based on the Bell Laboratories GR-Z graphics kernel [4] which is itself based on an even older graphics system [5]. It provides a device independent layer which is implemented in terms basic vector primitives, and an underlying set of device drivers which carry out the actual drawing.

*Department of Statistics, University of Auckland, New Zealand.

A good deal of low-level information on GR-Z is available in Becker and Chambers [2], and a descriptive overview is available in Chambers [4]. The author used this information together with some experience of implementing S device drivers as the basis for writing the system. Indeed, the R graphics device driver for the X window system is simply a modified version of an older S driver.

The GR-Z graphics system is now quite old and many of the devices it was originally designed to support are now no-longer in common use. Despite this, it must be regarded as one of the most successful systems of its type (other such systems include GKS and Core). Indeed, the quality of the graphics is often given as one of the most important reasons for using S. A good part of this success is almost certainly due to the fact that the graphics system was designed specifically to support graphics for data analysis and not just as a general graphics system.

In addition to the S graphics API, R has a number of extensions which users have found useful. In particular, the extension for adding mathematical annotation to plots (Murrell and Ihaka [8]) and a computational model for colour are finding good uses.

2 Key Graphics Features

In GR-Z (and hence in the R system) graphical output is produced as a series of one or more figures placed onto a single “page” of output. Each figure contains a central rectangular plot in which data are presented. The margins surrounding the plot contain additional annotation and labels. When multiple figures are placed on a page it is most common to arrange them as a one- or two-way array, but more general layouts are possible. In the case of R, Paul Murrell has implemented some significant additions to the layout facilities [6], [7].

Each figure contains a central plot region which usually contains the figure’s graphical display and is surrounded by margins. The plot region and its margins all have natural coordinate systems. The coordinates of the plot region are determined by the data display and the coordinates of the margin are those of the data display in one direction, and of lines of text in another. The use of margin coordinate systems makes it possible to provide a wide variety of annotation in a plot’s margins. This is a useful feature that even more modern graphics systems often do not provide.

It is possible to draw into the plot region and margins with a number of primitive vector graphical operations. S provides the operations:

<code>lines</code>	lines and polylines
<code>points</code>	glyph drawing
<code>polygons</code>	polygon drawing
<code>text</code>	text drawing

A special variant of the `text` primitive (named `mtext`) draws text into plot margins using the special coordinate systems for the margins. To these, R has added the special primitive `rectangles`, because it is such a commonly used operation which can benefit from a special implementation.

The S graphics system made the drawing operations device independent, but the implementers of device drivers were left free to use whichever set of colours and line textures they preferred. In R we have endeavoured to bring these features closer to true device independence. In particular, device drivers are passed 24 bit RGB colour specifications and are expected to do the best job they can in rendering these colours.

Since devices differ in how they render RGB, this is still not full device independence, but it does allow limited computations to be made on colours. In particular, it is possible to carry out shading computations based on lighting models. This has been used, for example, to add a small amount of visual realism to the `persp` function.

3 Graphics State

All graphics systems must deal with the problem of maintaining state. For example, POSTSCRIPT uses its `gsave` and `grestore` operators together with the ability to manipulate individual graphics state parameters to provide control over its operation. The GR-Z system has a very similar mechanism. At the interpreted level, the `par` function provides access to and control over the internal state of the graphics system. However, while POSTSCRIPT only provides relatively low-level capabilities, GR-Z provides a mix of low- and high-level capabilities. For example, the `par` function controls,

colour, line texture, line thickness, font family, font size, font rotation,
plotting symbol, plot type, axis type, tick length, margin size, figure
layout, . . .

This mixture of high- and low-level state parameters makes the `par` command the single most complicated S or R command. This complexity represents an obstacle which probably prevents many people from taking full advantage of the graphics system.

Although `par` controls a large number of graphics parameters, there is an argument that there is still too little state being maintained for good control of high-level plots. High-level level plots can contain a large number of primitive graphical elements (lines, points, text, etc.), and it may be desirable to control the properties of each of the elements separately.

There are two aspects to providing the user with control over the large number of graphics parameters needed to control a typical high-level display.

- (i) Controlling the default values for those parameters.
- (ii) Overriding the default values in a call to the function.

One means of providing this control is to expand the list of graphics parameters which `par` controls. As a result of experimentation in this direction, the present version of R has additional graphics parameters to control the marginal annotation

(main and sub-titles and axis labels) used in plots. In retrospect, the additional complexity that this has introduced seems to outweigh any benefits it has produced.

An alternative approach to packaging additional graphics control is to use the R closure mechanism to maintain the graphical parameters for a function “inside” that function. As a very simple illustrative example, the following code shows how to define an R function called `smoothplot` which plots a set of points and draws a smooth line through them.

```
(function() {  
  
  points.par <- list(pch = 1, col = "black")  
  
  lines.par <- list(lty = "solid", col = "black")  
  
smoothplot <-  
  function(x, y,  
           points.arg = list(),  
           lines.arg = list(),  
           setpar = FALSE,  
           resetpar = FALSE)  
  {  
    if (resetpar) {  
      points.par <- list(pch = 1, col = "black")  
      lines.par <- list(lty = "solid", col = "black")  
    }  
    if (setpar) {  
      points.par[names(points.arg)] <- points.arg  
      lines.par[names(lines.arg)] <- lines.arg  
    }  
    else {  
      points.par <- points.par  
      lines.par <- lines.par  
      points.par[names(points.arg)] <- points.arg  
      lines.par[names(lines.arg)] <- lines.arg  
    }  
  
    plot(x, y, type = "n")  
    points(x, y,  
          pch = points.par$pch,  
          col = points.par$col,  
          bg = points.par$bg)  
    lines(lowess(x, y),  
          lty = lines.par$lty,  
          col = lines.par$col)  
  }  
}  
})()
```

The code consists of a single expression which creates a function-closure that has access to an environment which contains variables `points.par` and `lines.par` which hold lists containing the graphics parameters associated with the points and lines in the plot.

The function can be used in a number of different ways. If invoked as

```
smoothplot(x, y)
```

the function plots the points and fitted smooth with the default parameters. These defaults can be overridden with function arguments as follows.

```
smoothplot(x, y,
           points = list(pch = 2, col = "red"),
           lines = list(col = "blue"))
```

The function can also be used to change the default graphics parameters for the plot as follows.

```
smoothplot(setpar = TRUE,
           points = list(col = "green4"),
           lines = list(col = "gray"))
```

Finally, it is possible to reset the parameters to their initial values.

```
smoothplot(resetpar = TRUE)
```

The use of closures to maintain graphics state provides a relatively simple extension to the standard `par` mechanism. The extension has the advantage that the state variables can be tailored to the particular graphical display being created. It also provides the advantage of providing tunable default parameters for purposes other than maintaining graphical state.

4 What The Graphics System Can Do

Because the R graphics system is defined in terms of primitive two-dimensional graphics operations, it is possible to draw a wide variety of graphs. Many common graphs have been packaged in a high-level way as R functions, but there is no reason for users to feel restricted to just these prepackaged forms. Using a little creativity and some elementary geometry it is possible to produce a variety of attractive graphical displays. In this section I'll show a few of the graphs I have produced with R for my introductory visualisation course.

Figure 1 is a redrawing of the classic map, created by Charles Joseph Minard and shown in Tufte [9]. The map shows the size of Napoleon's invading army during his 1812 campaign in Russia. The graph is made up of simple line, polygon, and text elements. The only difficulty in constructing this graph was obtaining the coordinates for the various elements in the graph. This was done by using the `xfig` drawing program to trace over the scanned outline of the graph and then extracting the coordinates from the resulting `xfig` data file.

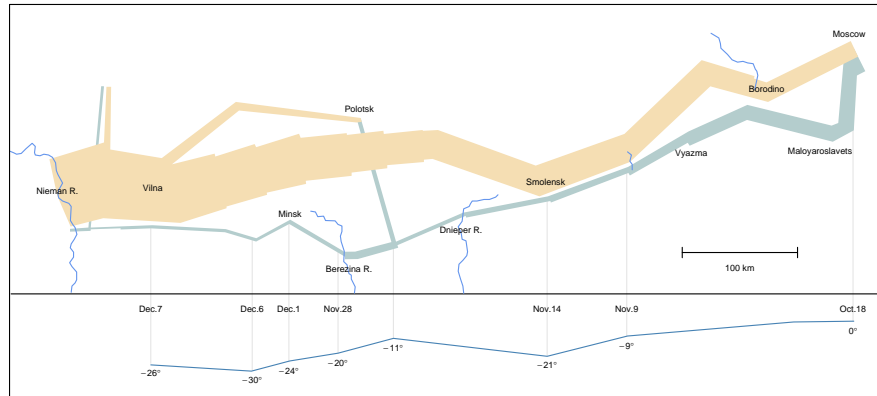


Figure 1: A redrawing of Charles Joseph Minard’s map of Napoleon’s 1812 campaign.

Figure 2 shows a plot of the results of the party vote for the 1999 New Zealand General Election in barycentric coordinates. The graph is of interest because New Zealand has recently changed its electoral procedure from a first-past-the-post one to one of proportional representation. The graph shows the bulk of the electorates falling in a narrow band along the political left-right axis (Labour is a centre-left party and National a centre right one). To the extreme left of the graph is a cluster of electorates formed by the Maori parliamentary seats and below that a single outlier which was the seat of a popular former Prime Minister.

Figure 3 shows an impression of the exterior of the RGB colour cube. The cube has been rendered in perspective, with hidden surfaces removed, and uses a computational model to generate the appropriate colours for the surface facets. The rendering was been carried out with a small experimental package of interpreted functions which extend the graphics system to three dimensions.

The three examples presented in this section represent relatively simple applications of the capabilities of the R graphics system. They show that it is possible to use the graphics system in a rather more creative way than is attempted by many users.

5 Future Work

Despite the fact that the graphics system contains enough functionality to produce a wide range of graphs there are some additional features which would clearly be useful.

- *More general polygon filling.* The present polygon primitive only deals with polygons defined by a single bounding contour. It would be useful to extend

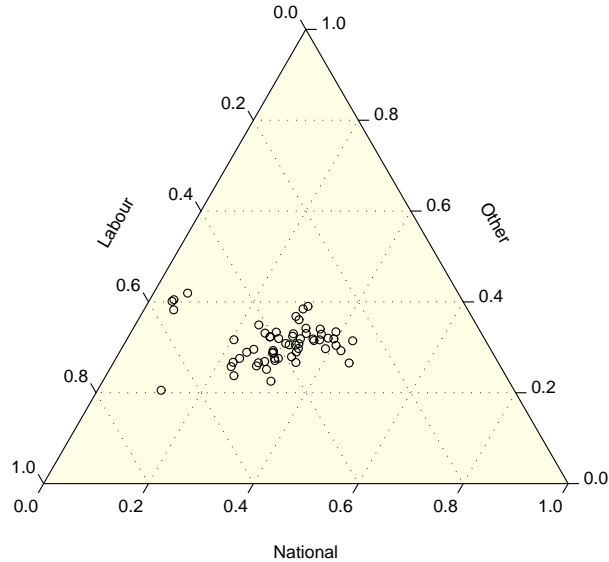


Figure 2: The per-electorate party vote for the 1999 New Zealand election.

the capability to polygons defined by multiple bounding contours. This would, for example, allow the filling of polygons “containing holes,” something which would be particularly useful for map drawing. It would be relatively simple to build this capability on top of the existing polygon facility.

- *Three dimensional capabilities.* The present graphics system is inherently two dimensional. It would be useful to add the ability to render three dimensional scenes in a simple way. The geometrical aspects of this are relative straightforward, but to be truly useful hidden line and surface capabilities are needed. Because the underlying primitives are vector ones, many of the current computer graphics methods are not applicable. There are some techniques such as those based on BSP trees which could be used.
- *Raster capabilities.* If it were possible for device drivers to use the underlying raster capabilities which most devices possess, it would be possible to add some relative nice features such as smooth (Phong or Gouraud) shading for three dimensional graphs, or to use raster techniques to handle hidden line and surface problems.
- *An improved font system.* Current font technology makes it virtually impossible to produce truly device independent text in graphs. Some progress is being made in this area and it may well be worth revisiting the text capabilities of the current R graphics system to see if current developments are useful.

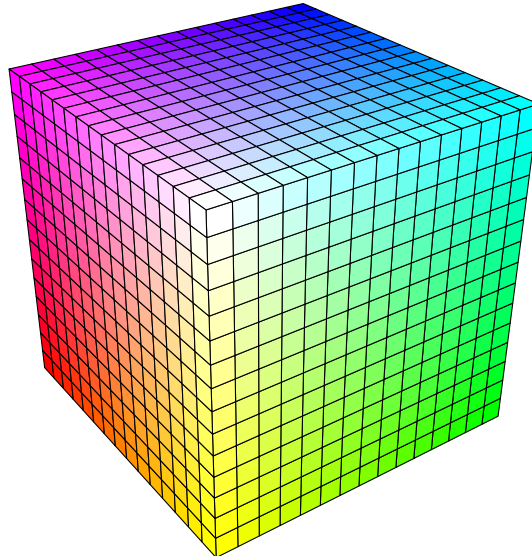


Figure 3: The RGB colour cube.

All of these extensions would be relatively straightforward to implement by building on the existing graphics capabilities or by adding simple new graphics primitives.

Another interesting question is whether it might be possible to include completely new graphics systems alongside or in place of the existing one. One approach to this would be to fully separate out the device dependent functionality of the present system from the device independent ones. This would allow a new system to target the existing devices.

6 Conclusions

The R graphics system is based on relatively old and stable technology. It is clear however that there are still gains in usability and functionality to be made. The difficulty in making the changes is primarily that of providing backward compatibility. Even with this problem, small changes of the type outlined in this paper can still lead to improved functionality.

References

- [1] R. A. Becker and J. M. Chambers. *S. An Interactive Environment for Data Analysis and Graphics*. Wadsworth and Brooks/Cole, Monterey, 1984.

- [2] R. A. Becker and J. M. Chambers. *Extending the S System*. Wadsworth and Brooks/Cole, Monterey, 1985.
- [3] R. A. Becker, J. M. Chambers, and A. R. Wilks. *The NEW S Language*. Wadsworth and Brooks/Cole, Monterey, 1988.
- [4] J. M. Chambers. *Computational Methods for Data Analysis*. Wiley, New York, 1977.
- [5] J. M. Chambers. Personal communication. 2001.
- [6] P. R. Murrell. Layouts: A mechanism for arranging plots on a page. *JCGS*, 8:121–134, 1999.
- [7] P. R. Murrell. R lattice graphics. In K. Hornik and F. Leisch, editors, *DSC 2001 Proceedings of the 2nd International Workshop on Distributed Statistical Computing*, Vienna, Austria, March 15-17 2001. ISSN 1609-395X.
- [8] P. R. Murrell and R. Ihaka. An approach to providing mathematical annotation in graphs. *JCGS*, 8:582–599, 2000.
- [9] E. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, Connecticut, 1986.