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*

RGL: An R Interface to OpenGL

Duncan Murdoch*

Abstract

OpenGL is a cross-platform library for 3D rendering. In this paper I will describe RGL, an R package providing a simplified interface to it. The style of the interface follows that of the older S or R plotting functions, with some differences. In particular, it is possible for the user to manipulate objects on the screen, and programmatically to change the appearance or content of the display.

1 Introduction

OpenGL [5] is a hardware-independent library of about 150 functions for 3D graphics rendering. Implementations exist on all of the more popular computing platforms in use today: MS Windows, various Unix versions, Apple Macintosh, etc. It is a direct descendant of the GL graphics library written by Silicon Graphics. Most recent 3D video hardware comes with OpenGL drivers, allowing programs written for OpenGL to automatically take advantage of hardware acceleration features.

R [3] is a GNU implementation of the S statistical programming language [1]. In this paper I describe a prototype implementation of RGL, an R package providing a simplified interface to OpenGL. I will assume in this paper that readers are familiar with R and S, but perhaps not with OpenGL.

In the remainder of this section I give a short overview of OpenGL and Delphi, the package used to compile the external code in RGL.

*University of Western Ontario

1.1 OpenGL

OpenGL is a complex library. There is support for drawing 3 dimensional geometric primitives (points, lines and polygons). It automatically handles depth cues such as perspective, hiding objects behind ones that are closer to the viewer, fog that makes more distant objects tend to fade out, etc.

Light in OpenGL may come from ambient sources (the same brightness in all directions), diffuse sources (greater brightness from some general directions), or point sources. The viewer sees the effect of lighting on surfaces defined using different *materials*. A material is defined by its reflectivity: it may have different ambient, diffuse and specular (shiny) properties. Materials may also be light sources themselves, or may be transparent.

A number of tricks are available to improve the three dimensional illusion while using simple polygon tilings. *Textures*, or bitmapped patterns, may be displayed on surfaces to give the illusion of more detail than is really being drawn. Colours are defined at polygon vertices, and may be smoothly blended over the surface of the polygon. In order to know the reflectivity of a surface, a normal needs to be defined, but the normal need not be the actual geometric normal to the polygon: indeed, different normals can be used at each vertex, with values in the interior of a polygon interpolated to simulate a curved surface.

Most of OpenGL is designed to be platform independent, but some necessary features are left out of the basic library, and are implemented in a platform dependent way. These include interaction with the computer's windowing system, and the display of text.

1.2 Delphi

The interface to OpenGL is described in the C language, but no C-specific constructs are required, so OpenGL is callable from many other languages. The prototype implementation of RGL described in this paper was programmed in Delphi [2]. Delphi is a component-based programming package [4], which allows easy prototyping of the user interface. The most natural way to add external code to R is through a shared library (a DLL in Windows); Delphi was used to compile this library.

It is my intention to rewrite the shared library in C, once the design of RGL has stabilized. This will make it possible to port RGL to other platforms besides Windows.

2 RGL Overview

Whereas OpenGL is a complex low level graphics library, RGL is designed to offer both high level and low level support, similar in many respects to the graphics model defined in R. See Table 1 for a list of most of the RGL functions.

R has functions `plot`, `hist` and `persp` for scatterplots, histograms and perspective plots respectively; RGL has functions `plot3d`, `hist3d` and `persp3d`. The options to the RGL functions are very similar to the options to the default versions of the R functions. For example, Figure 1 shows the result of the R code

High level	`hist3d, persp3d, plot3d`
Low level	`lines3d, segments3d, points3d, triangles3d, quads3d, text3d`
Annotations	`axis3d, axes3d, box3d, mtext3d, title3d`
Parameters	`par3d`
Windows	`open3d, close3d, clear3d`
Colours	`rgb.to.color, color.to.rgb`
Transforms	`translate3d, rotate3d, scale3d`
Modifiers	`setcolors3d, setnormals3d, setpoints3d, settext3d`
Grouping	`begingroup3d, endgroup3d`
Objects	`print.obj3d, summary.obj3d, as.obj3d, as.list.obj3d, [.obj3d, type3d, length3d, items3d`

Table 1: The main functions defined in RGL.

```
x <- rnorm(1000)
y <- rnorm(1000)
hist3d(x,y,col='red')
```

At the lower level, RGL has functions `points3d, lines3d, segments3d, text3d, triangles3d` and `quads3d`. The first four of these correspond to similarly named functions in R; the latter two draw filled three dimensional polygons. These functions are used to build up the higher level graphics.

At an intermediate level, the functions `axis3d, axes3d, box3d, mtext3d` and `title3d` are used to label and annotate graphs.

Like R, RGL has a function `par3d` for setting graphical parameters. About 25 parameters can be set or queried. These include parameters for scaling (RGL can rescale each coordinate separately, or apply the same scaling to each), parameters describing the position and colour of the graphics window, OpenGL display parameters controlling lighting, fog and perspective, and parameters controlling the thickness of lines and points. There are also parameters controlling the interaction of the user with the display. By default, the user is only allowed to rotate the display about a vertical axis (since it is easy to get disoriented when the up-down direction is changed), but `par3d` allows more freedom to be granted. Other parameters tell RGL to skip updates of the display (so that a number of new objects can be added without redraws after each one) or to ignore the extent of objects when calculating scaling (so that axes and labels on a graph don't affect its scaling).

Besides these functions that correspond to functions in R, RGL provides several other groups of functions. There are functions for opening, closing and clearing display windows, functions for translating colours from the system used in R to a numeric Red-Green-Blue system, and functions for creating transformation matrices. OpenGL normally works in four dimensional homogeneous coordinates in which vectors (x, y, z, w) are used to represent the point $(x/w, y/w, z/w)$; RGL supplies functions to construct 4×4 matrices representing translations, rotations, and rescalings of the data. (A quick introduction to homogeneous coordinates is given

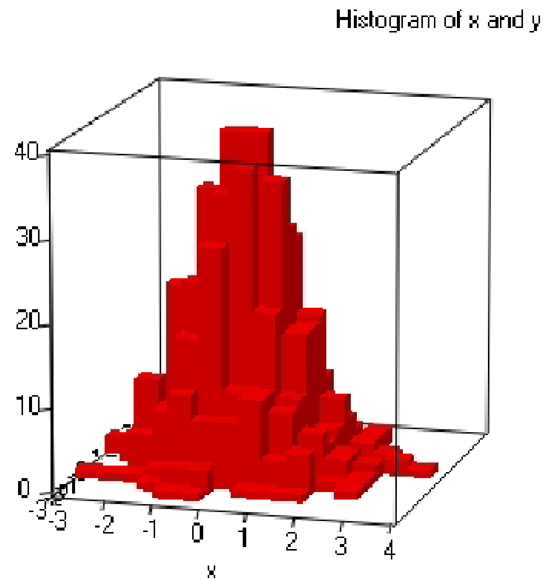

Figure 1: A simple bivariate histogram.

in an appendix to the OpenGL reference manual [5].)

The "modifier" group of functions allow the data in a plot to be changed by the user. The graphics model in R is essentially "drawing in ink": once objects are placed on a graph, they may be covered by later objects, but they can't be changed or deleted. This allows it to work directly with printers and other hard copy output devices. RGL, on the other hand, is essentially dynamic: in order to present different views of three dimensional objects, it's necessary to redraw them many times per second. RGL allows the user to change what is redrawn, by giving access to the internal data being displayed in the graph. Internally data is maintained in the original scale; functions are available to query and change the values. It is also possible to change the normals and colours at each point, and the strings used for text.

RGL defines graphical scenes hierarchically. Each plot is a group of objects defined in a particular frame of reference. Some of the objects may be groups themselves, with their own internal coordinate systems. To support this model, there are the functions `begingroup3d` which declares that a new group is being drawn, and `endgroup3d` which embeds it in a larger scene.

Finally, a number of functions treat RGL objects as R objects. All of the data in the RGL objects is maintained externally, in a format suitable for OpenGL display; only "handles" (currently implemented as pointers typecast to integers) to the external objects are seen in R. However, functions that create these handles give them the class `obj3d`, and support functions to print and work with this class

are defined, so that in many respects it appears to the R user as though the objects are fully defined within R.

3 Example

The `plot3d` function operates much like R's `plot` function, and on paper, the displays don't look very three dimensional. In this section I'll present an example using `persp3d` instead.

Consider Ross Ihaka's volcano data from the R distribution. To draw this in perspective form, the following modification of the example code for `persp` could be used:

```
> data(volcano)
> z <- 2 * volcano        # Exaggerate the relief
> x <- 10 * (1:nrow(z))   # 10 meter spacing (S to N)
> y <- 10 * (1:ncol(z))   # 10 meter spacing (E to W)
> p <- persp3d(x, y, z, scale = FALSE, axes = FALSE,
+               smooth = T, xlab = '', ylab = '', zlab = '')
```

This produces results very similar to what `persp` would produce, but the surface is smooth (the normals are fit separarately at every vertex) and the user can rotate the image to get a better view.

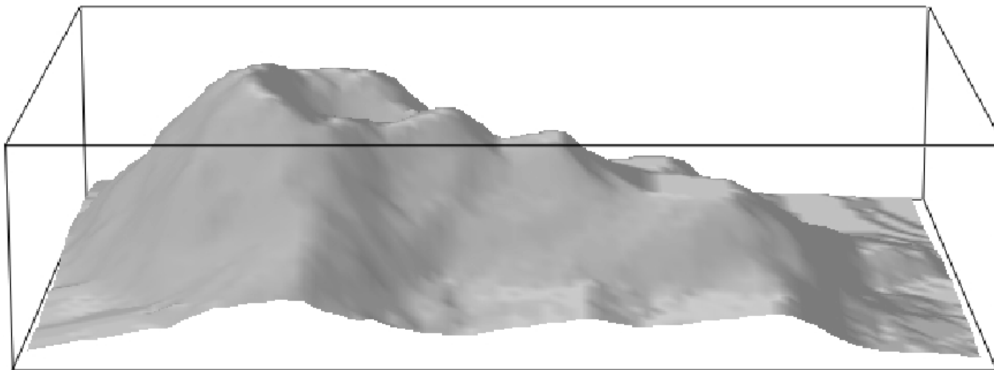

Figure 2: The `volcano` dataset displayed by `persp3d`.

We might choose to colour the volcano depending on its altitude. The `persp3d` function has two arguments to support this: `col` and `rgb`. The `col` argument takes colours specified in the usual way with R, i.e. as selections from a palette or as text strings naming the colour. The `rgb` argument (with default value `rgb = color.to.rgb(col)`) works with the numerical RGB colour values instead. Colour can be given uniformly for the surface, or separately for each point in the grid of z

values. It is generally faster to specify the colours using the `rgb` argument, because fewer conversions to the numerical format will then be required.

Another way to colour the graph is to use the modifier functions on the existing plot. The variable `p` that stores the result of the `persp3d` call has four members:

```
> names(p)
[1] "handle" "indx"   "indy"   "indz"
```

The `handle` member is the handle of the list of triangles that make up the plotted surface. The other three members are lists of indices into the original `x, y` and `z` variables corresponding to the vertices of the triangles. These allow us to modify the graph without redrawing it:

```
> r <- range(z)
> zvect <- (z[p$indz] - r[1])/(r[2]-r[1])
> palette <- color.to.rgb(terrain.colors(50))
> rgb <- palette[round(zvect*49,0)+1]
> setcolors3d(p$handle, rgb = rgb)
```

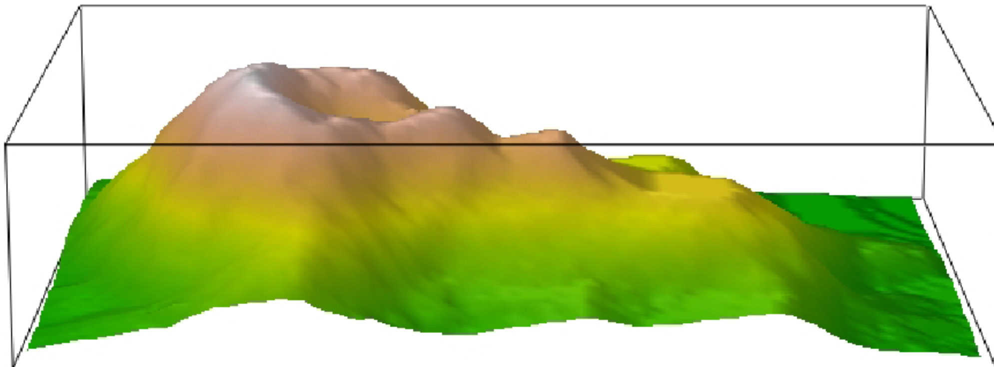

Figure 3: The `volcano` dataset with the colours modified to indicate altitude.

First we calculate a vector `zvect` containing the standardized `z` values at every vertex on the drawn surface. We then use the `terrain.colors` function to generate a sequence of 50 colours, and convert them to the numerical RGL format. Next we create a vector `rgb` containing the colour corresponding to the altitude at each vertex. Finally, the `setcolors3d` call changes the displayed surface so that it uses these colours (Figure 3).

4 Discussion

RGL is still in the prototype stage. Many OpenGL features are not presently supported: multiple lights, changes to the surface material properties, textures,

the more exotic drawing modes, font selection, etc. I do have short-term plans to support materials and font selection; the other items will have to wait.

I think RGL demonstrates some ideas that may be applicable to a new version of R graphics in general. In particular, the ability to modify data in a plot makes many things possible: animations, sequential displays which automatically rescale, etc.

RGL is currently available from my web page

`http://www.stats.uwo.ca/faculty/murdoch/software`

but only in binary form for the Windows version of R. Once the design has settled down, I plan to convert the external library code (about 3700 lines in Delphi) to C, so that it will be more widely available.

References

[1] John M. Chambers and Trevor J. Hastie. *Statistical Models in S.* Chapman & Hall, 1992.

[2] Borland Software Corporation. *Borland Delphi Professional Version 5.0.* 1999.

[3] Ross Ihaka and Robert Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996.

[4] Duncan J. Murdoch. Programming with components. *Chance*, 12(4):47–49, 1999.

[5] Mason Woo, Jackie Neider, and Tom Davis. *OpenGL Programming Guide, Second Edition.* Addison-Wesley Developers Press, 1997.