# The gridSVG Package

*by Paul Murrell and Simon Potter*

**Abstract** The **gridSVG** package can be used to generate a **grid**-based R plot in an SVG format, with the ability to add special effects to the plot. The special effects include animation, interactivity, and advanced graphical features, such as masks and filters. This article provides a basic introduction to important functions in the **gridSVG** package and discusses the advantages and disadvantages of **gridSVG** compared to similar R packages.

## Introduction

The SVG graphics format (Dengler et al., 2011) is a good format for including plots in web pages because it is a vector format (so it scales well) and because it offers features for animation and interactivity. It is possible to produce a static R plot in an SVG format with the built-in svg() function (from the **grDevices** package), but the **gridSVG** package (Murrell and Potter) provides an alternative way to generate an SVG plot that allows for creating animated and interactive graphics.

This article demonstrates basic usage of the **gridSVG** package and outlines some of the ways that **gridSVG** can be used to produce graphical results that are not possible in standard R graphics. There is also a discussion of other packages that provide ways to generate dynamic and interactive graphics for the web and the strengths and weaknesses of **gridSVG** compared to those packages.

## Basic usage

As the package name suggests, the **gridSVG** package starts with a plot that was drawn using the **grid** graphics package, or a package built on top of **grid**, like **lattice** (Sarkar, 2008) or **ggplot2** (Wickham, 2009). Figure 1 shows an example plot, on a standard R graphics device, that was produced using the **osmar** and **sp** packages (Eugster and Schlesinger, 2010; Pebesma and Bivand, 2005). The **sp** function spplot() uses **lattice** (which uses **grid**) for drawing.

The grid.export() function in **gridSVG** converts the current (**grid**) scene on the active graphics device to an SVG format in an external file. The SVG file can be viewed directly in a browser (see Figure 2) or embedded within HTML as part of a larger web page.

This usage of **gridSVG**, to produce a static SVG version of an R plot for use on the web, offers no obvious benefit compared to the built-in svg() graphics device. However, the **gridSVG** package provides several other functions that can be used to enhance the SVG version of an R plot.

## A simple example

In order to demonstrate, with code, some of the distinctive features of **gridSVG**, we introduce a simple **grid** scene that is inspired by the Monty Hall problem.[1]

```
> library(grid)
```

The scene consists of three words, "goat", "goat", and "car", drawn in random order across the page, with an opaque rectangle drawn on top of each word.

In relation to the Monty Hall problem, the three rectangles represent three "doors", behind which are hidden two goats and a car. A "contestant" must choose a door and then he or she gets the "prize" behind that door. However, after the contestant has chosen a door, a "game show host" opens one of the other doors to reveal a "goat" and the contestant gets the opportunity to change to the remaining unopened door or stick with the original choice. Should the contestant stick or switch?[2]

The following code produces the scene and the result is shown in Figure 3. The main drawing code is wrapped up in a function so that we can reuse it later on.

```
> text <- sample(c("goat", "goat", "car"))
> cols <- hcl(c(0, 120, 240), 80, 80)
```

---

[1] http://en.wikipedia.org/wiki/Monty_Hall_problem

[2] An exercise for the reader is to determine which door conceals the car based on the R code and figures presented in this article.
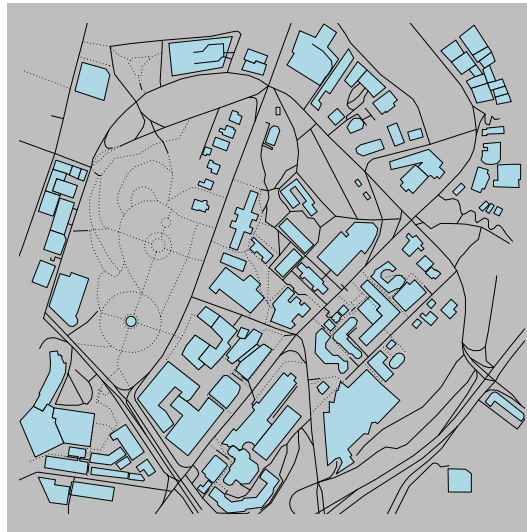
**Figure 1:** A map showing buildings, streets, and paths in the region around The University of Auckland campus. The map was produced using the **osmar** and **sp** packages (and **lattice** and **grid** for the drawing) on the standard pdf() graphics device in R.
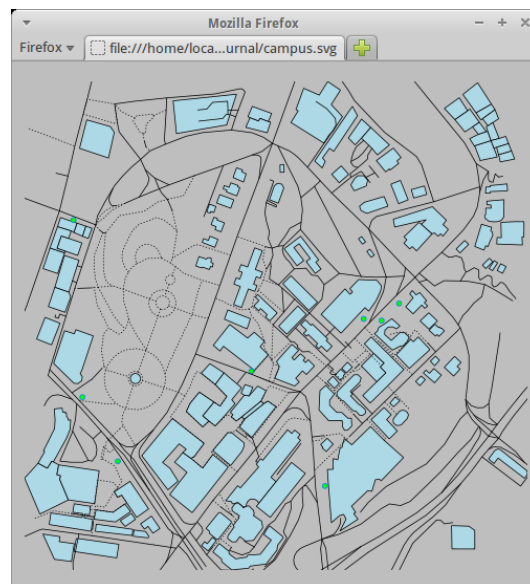


**Figure 2:** The map from Figure 1 exported to an SVG file by **gridSVG** and viewed in Firefox. This demonstrates that a static R plot can be converted to an SVG format with **gridSVG** for use on the web.
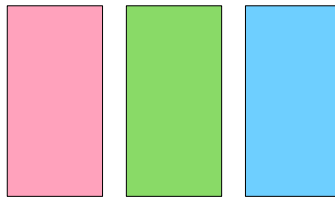
**Figure 3:** A diagram of the Monty Hall problem, drawn using **grid**. Hidden behind each rectangle is either the word "goat" or the word "car").

```
> MontyHall <- function() {
      grid.newpage()
      grid.text(text, 1:3/4, gp=gpar(cex=2), name="prizes")
      for (i in 1:3) {
          grid.rect(i/4 - .1, width=.2, height=.8, just="left",
                    gp=gpar(fill=cols[i]), name=paste0("door", i))
      }
  }

> MontyHall()
```

The three rectangles in this scene have been given names—"door1", "door2", and "door3"—as shown in the output from **grid**'s grid.ls() function below.

```
> grid.ls()

text[prizes]
rect[door1]
rect[door2]
rect[door3]
```

These names will be used later to identify the rectangles so that we can modify them to generate special effects.

## Hyperlinks

The grid.hyperlink() function from the **gridSVG** package can be used to add hyperlinks to parts of a **grid** scene. For example, the following code adds a link to each door so that clicking on a door (while viewing the SVG version of the scene in a browser) leads to a Google Image Search on either "car" or "goat" depending on what is behind the door. The first argument to grid.hyperlink() is the name of the **grid** object with which to associate the hyperlink. The href argument provides the actual link.

```
> library(gridSVG)

> links <- c("http://www.google.com/search?q=car&tbm=isch",
             "http://www.google.com/search?q=goat&tbm=isch")
> for (i in 1:3) {
      grid.hyperlink(paste0("door", i),
                     href=links[match(text[i], c("car", "goat"))])
  }
```

After running this code, the scene is completely unchanged on a normal graphics device, but if we use grid.export() to convert the scene to SVG, we end up with an image that contains hyperlinks. Figure 4 shows the result, with the mouse hovering over the middle door; at the bottom-left of the browser window, we can see from the hyperlink that there is a goat behind this door.

```
> grid.export("montyhall-hyper.svg")
```
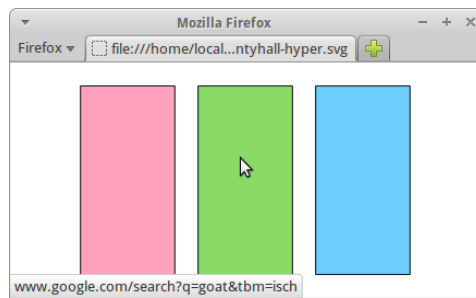
**Figure 4:** The Monty Hall image, with a hyperlink on each door. The mouse is hovering over the middle door and the browser is showing the hyperlink target in the bottom-left of its window. If we click the mouse, we will navigate to a Google Image Search for the word "goat".
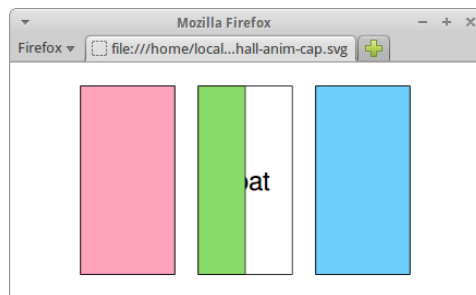


**Figure 5:** The Monty Hall image, with the middle "door" animated so that it slides open (to reveal the word "goat").

## Animation

The function `grid.animate()` allows us to animate the features of shapes in a **grid** scene. For example, the following code draws the Monty Hall scene again and then animates the width of the middle door so that it slides open (to reveal the word "goat"). The first argument to `grid.animate()` is the name of the object to animate. Subsequent arguments specify which feature of the object to animate, in this case `width`, plus the values for the animation. The `duration` argument controls how long the animation will last.

```
> MontyHall()
> goatDoor <- grep("goat", text)[1]
> grid.animate(paste0("door", goatDoor), width=c(.2, 0), duration=2)

> grid.export("montyhall-anim.svg")
```

Again, no change is visible on a normal R graphics device, but if we export to SVG and view the result in a browser, we see the animation (see Figure 5).

## Advanced graphics features

The **gridSVG** package offers several graphics features that are not available in standard R graphics devices. These include non-rectangular clipping paths, masks, fill patterns and fill gradients, and filters (Murrell and Potter, 2013). This section demonstrates the use of a mask on the Monty Hall scene.

A mask is a greyscale image that is used to affect the transparency (or alpha-channel) of another image: anywhere the mask is white, the masked image is fully visible; anywhere the mask is black, the masked image is invisible; and anywhere the mask is grey, the masked image is semitransparent.

The following code defines a simple scene consisting of a white cross on top of a grey circle on a white background, which we will use as a mask (see Figure 6). Any **grid** scene can be used to create a mask.

```
> circleMask <- gTree(children=gList(rectGrob(gp=gpar(col=NA, fill="white")),
                      circleGrob(x=goatDoor/4, r=.15,
```

**Figure 6:** Using a mask on an image. The picture on the left shows a white cross on top of a grey circle on a white background. This is used as a mask on the rectangles in the Monty Hall image on the right. The effect is to create a semitransparent window in the middle door (through which we can glimpse the word "goat").

```
                        gp=gpar(col=NA, fill="grey")),
              polylineGrob(c(0, 1, .5, .5),
                           c(.5, .5, 0, 1),
                           id=rep(1:2, each=2),
                           gp=gpar(lwd=10, col="white"))))
```

The next code shows how this crossed circle on a white background can be used as a mask to affect the transparency of one of the rectangles in the Monty Hall scene. The first argument to `grid.mask()` is the name of the object to mask and the second argument is the mask object (as created by the `mask()` function).

```
> MontyHall()
> grid.mask(paste0("door", goatDoor), mask(circleMask))

> grid.export("montyhall-masked.svg")
```

The effect (in the exported SVG image) is to create a little window in the middle door (to reveal what looks suspiciously like the word "goat"; see Figure 6).

## Interactivity

The `grid.garnish()` function in the **gridSVG** package opens up a broad range of possibilities for enhancing a **grid** scene, particularly for adding interactivity to the scene.

A simple example is shown in the code below. Here we are adding tooltips to each of the doors in the Monty Hall scene so that hovering the mouse over a door produces a label that shows what is behind the door (see Figure 7). The first argument to `grid.garnish()` is the name of the object to modify. Subsequent arguments specify SVG attributes to add to the object; in this case, we add a `title` attribute, which results in a tooltip (in most browsers).

```
> MontyHall()
> for (i in 1:3) {
      grid.garnish(paste0("door", i), title=text[i])
  }

> grid.export("montyhall-tooltip.svg")
```

The `grid.garnish()` function can also be used to associate JavaScript code with an object in the scene. The following code shows a simple example where clicking on one of the rectangles pops up an alert box showing what is behind that door (see Figure 8). The attribute in this example is `onclick`, which is used to define an action that occurs when the object is clicked with the mouse (in a browser).

```
> MontyHall()
> for (i in 1:3) {
      grid.garnish(paste0("door", i),
                   onclick=paste("alert('", text[i], "')"))
  }
```

**Figure 7:** The Monty Hall image with tooltips added to each door. The mouse is hovering over the middle door, which results in a tooltip being displayed to show that there is a "goat" behind this door.
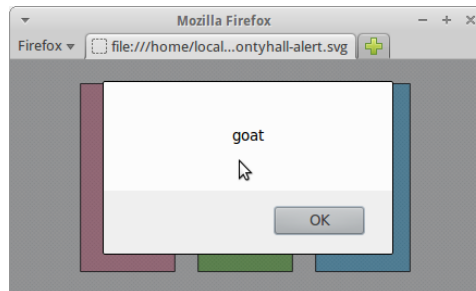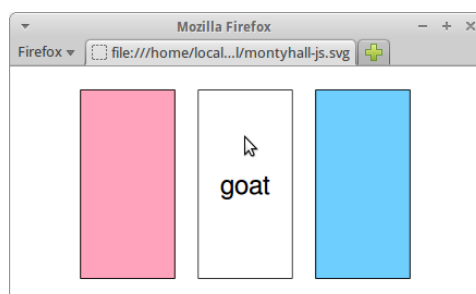


**Figure 8:** The Monty Hall image with interactivity. The mouse has just been clicked on the middle door, which has resulted in an alert box popping up to show that this door has a "goat" behind it.

```
> grid.export("montyhall-alert.svg")
```

For more complex interactions, it is possible to include JavaScript code within the scene, using the grid.script() function, so that an event on an object within the scene can be associated with a JavaScript function call to perform a more sophisticated action. The code below shows a simple example where clicking on one of the rectangles in the Monty Hall scene will call the JavaScript function open() to "open" the door (by making the rectangle invisible; see Figure 9). The open() function is defined in a separate file called "MontyHall.js" (shown in Figure 10).

```
> MontyHall()
> for (i in 1:3) {
      grid.garnish(paste0("door", i), onclick="open(evt)")
  }
> grid.script(file="MontyHall.js")

> grid.export("montyhall-js.svg")
```



**Figure 9:** The Monty Hall image with more interactivity. The mouse has just been clicked on the middle door, which has resulted in the middle door becoming invisible, thereby revealing a "goat" behind the door.

```
open = function(e) {
    e.target.setAttribute("visibility", "hidden");
}
```

**Figure 10:** The JavaScript code used in Figure 9 that defines the open() function to "open" a door by making the rectangle invisible.
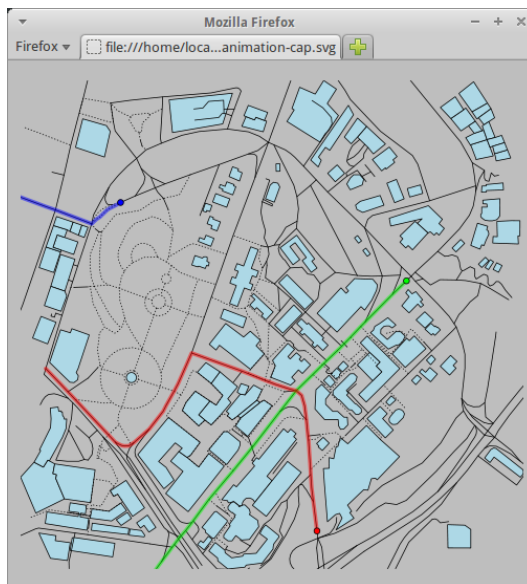


**Figure 11:** The map from Figure 1 with animated lines added to show several bus routes around The University of Auckland campus.

## A more complex demonstration

The previous section kept things very simple in order to be able to show all of the R code involved in the examples. In this section, we present some more complex examples. The R code is not shown, but is available online, along with "live" versions of the images (links are given at the end of this article).

Figure 11 shows the campus map from Figure 1 with some animations added in the form of blue, green and red lines that describe three inner-city bus routes that pass through The University of Auckland campus. This demonstrates the use of grid.animate() with a much more complex **grid** scene.

Figure 12 shows a mask being applied to the campus map from Figure 1. On the left, the mask consists of a complex shape that defines the extent of The University of Auckland campus as white regions on a dark grey background. On the right is the result of using that mask on the campus map; all regions outside the extent of the campus are faint because the dark grey areas on the mask result in semitransparency for those regions.

## Limitations

The **gridSVG** package provides an opportunity to produce more sophisticated, more dynamic, and more interactive R plots compared to the standard R graphics devices. However, there are some strict limitations on what can be achieved with this package.

First of all, the package only works for plots that are based on the **grid** graphics system. This includes some major graphics packages, such as **lattice** and **ggplot2**, but excludes a large amount of graphics functionality that is only available in the default **graphics** package or packages that build on **graphics**.

Another major limitation is that **gridSVG** does not generate any JavaScript code itself. This means that anything beyond the most basic interactivity will require the user to write JavaScript code, which imposes a burden on the user in terms of both time and knowledge.

Another point that has only briefly been acknowledged in the example R code so far is that the
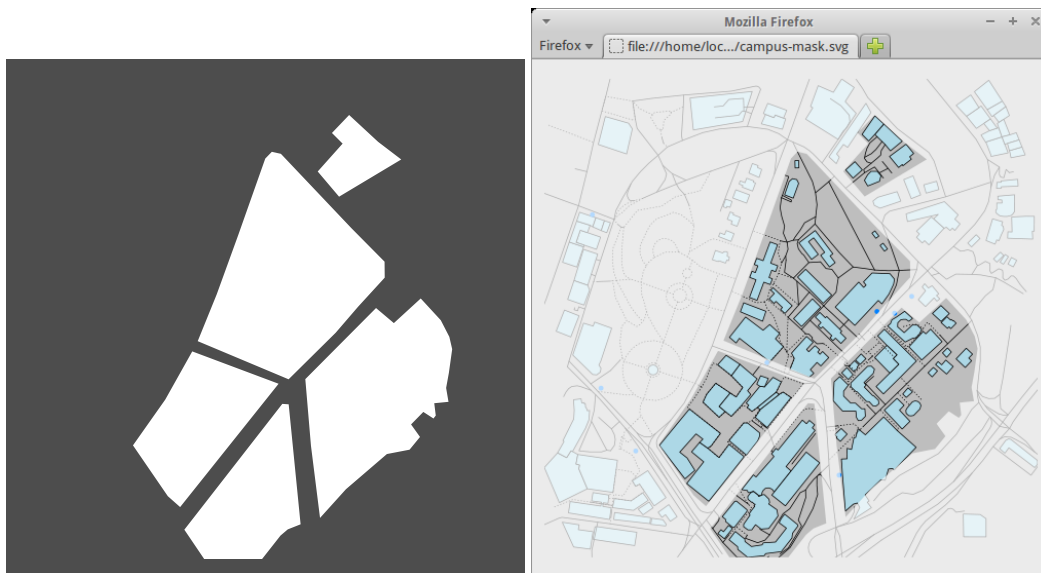
**Figure 12:** The map from Figure 1 (right) with a mask (left) used to de-emphasize areas of the map that are not part of The University of Auckland campus.

**gridSVG** functions that add special features to a **grid** scene (such as hyperlinks and animation) rely heavily on the ability to *identify* specific components of a **grid** scene. The Monty Hall examples all rely on the fact that the rectangles that are drawn to represent doors each have a name—"door1", "door2", and "door3"—and the code that adds hyperlinks or animation identifies the rectangles by using these names. This means that **gridSVG** is dependent upon an appropriate naming scheme being used for any **grid** drawing (Murrell, 2012). This requirement is met by the **lattice** package and, to a lesser extent by the **ggplot2** package, but cannot be relied on in general.

## Alternative approaches

The **gridSVG** package provides one way to produce dynamic and interactive versions of R plots for use on the web, but there are several other packages that provide alternative routes to the same destination. This section discusses the differences between **gridSVG** and several other packages that have similar goals.

The **animation** package (Xie, 2013) provides a convenient front-end for producing animations in various formats (some of which are appropriate for use on the web), but the approach is frame-based (draw lots of separate images and then stitch them together to make an animation). The advantage of an SVG-based approach to animation is that the animation is declarative, which means that the animation can be described more succinctly and efficiently and the resulting animation will often appear smoother.

The **SVGAnnotation** package (Nolan and Lang, 2012) performs a very similar role to **gridSVG**, by providing functions to export R plots to an SVG format with the possiblity of adding dynamic and interactive features. One major advantage of **SVGAnnotation** is that it will export R plots that are based the standard **graphics** package (as well as plots that are based on **grid**). **SVGAnnotation** also provides some higher-level functions that automatically generate JavaScript code to implement specific sorts of more complex interactivity. For example, the linkPlots() function can be used to generate linked plots, where moving the mouse over a data symbol in one plot automatically highlights a corresponding point in another plot. The main disadvantage of **SVGAnnotation** is that it works with the SVG that is produced by the built-in svg() device, which is much less structured than the SVG that is generated by **gridSVG**. That is not a problem if the functions that **SVGAnnotation** provides do everything that we need, but it makes for much more work if we need to, for example, write our own JavaScript code to work with the SVG that **SVGAnnotation** has generated.

A number of packages, including **rCharts** and **googleVis** (Vaidyanathan, 2013; Gesmann and de Castillo, 2011), provide a quite different approach to producing dynamic and interactive plots for the web. These packages outsource the plot drawing to JavaScript libraries such as NVD3, highcharts, and the Google Visualisation API (Novus, 2012; Highsoft AS, 2013; Google, 2013). The difference here is that the plots produced are not R plots. The advantage is that very little R code is required

to produce a nice result, provided the JavaScript library can produce the style of plot and the sort of interactivity that we want.

Another approach to interactivity that is implemented in several packages, notably **shiny** (RStudio Inc., 2013), is running R as a web server and producing new R graphics in response to user events in the browser. The difference here is that the user typically interacts with GUI widgets (buttons and menus) outside the graphic and each user event generates a completely new R graphic. With **gridSVG**, the user can interact directly with elements of the graphic itself and all of the changes to the graphic occur in the browser with no further need of R.

In summary, using the **gridSVG** package is appropriate if we want to add advanced graphics features to a **grid**-based R plot, or if we want to add dynamic or interactive elements to a **grid**-based R plot, particularly if we want to produce a result that is not already provided by a high-level function in the **SVGAnnotation** package. An approach that holds some promise is to generate SVG content using **gridSVG** and then manipulate that content by adding JavaScript code based on a sophisticated JavaScript library such as d3 (Bostock et al., 2011).

## Availability

The **gridSVG** package is available from CRAN. The code examples are known to work for **gridSVG** versions 1.3 and 1.4.

Online versions of the figures in this article, along with complete code for the campus map examples, are available from http://www.stat.auckland.ac.nz/~paul/Reports/gridSVGrj/.

Further documentation and examples for **gridSVG** are available from https://www.stat.auckland.ac.nz/~paul/R/gridSVG/.

## Bibliography

M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-Driven Documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2011. URL http://vis.stanford.edu/papers/d3. [p9]

P. Dengler, D. Jackson, C. Lilley, J. Fujisawa, C. McCormack, E. Dahlström, A. Grasso, J. Ferraiolo, D. Schepers, and J. Watt. Scalable vector graphics (SVG) 1.1 (second edition). W3C recommendation, W3C, Aug. 2011. http://www.w3.org/TR/2011/REC-SVG11-20110816/. [p1]

M. J. A. Eugster and T. Schlesinger. osmar: Openstreetmap and r. *R Journal*, 2010. URL http://osmar.r-forge.r-project.org/RJpreprint.pdf. Accepted for publication on 2012-08-14. [p1]

M. Gesmann and D. de Castillo. googleVis: Interface between R and the Google Visualisation API. *The R Journal*, 3(2):40–44, December 2011. URL http://journal.r-project.org/archive/2011-2/RJournal_2011-2_Gesmann+de~Castillo.pdf. [p8]

Google. Google Visualization API, 2013. URL https://developers.google.com/chart/interactive/docs/reference. [p8]

Highsoft AS. Highcharts JS, 2013. URL http://www.highcharts.com/. [p8]

P. Murrell. What's in a Name? . *The R Journal*, 4(2):5–12, dec 2012. URL http://journal.r-project.org/archive/2012-2/RJournal_2012-2_Murrell.pdf. [p8]

P. Murrell and S. Potter. *gridSVG: Export grid graphics as SVG*. R package version 1.4-0. [p1]

P. Murrell and S. Potter. Advanced SVG Graphics from R. Technical Report 2013-7, Department of Statistics, The University of Auckland, 2013. URL http://stattech.wordpress.fos.auckland.ac.nz/2013-7-advanced-svg-graphics-from-r/. [p4]

D. Nolan and D. T. Lang. Interactive and animated scalable vector graphics and r data displays. *Journal of Statistical Software*, 46(1):1–88, 1 2012. ISSN 1548-7660. URL http://www.jstatsoft.org/v46/i01. [p8]

Novus. NVD3.js : Re-usable charts for d3.js, 2012. URL http://nvd3.org/. [p8]

E. Pebesma and R. Bivand. Classes and methods for spatial data in r. *R News*, 2, 2005. URL http://cran.r-project.org/doc/Rnews/. [p1]

RStudio Inc. *shiny: Web Application Framework for R*, 2013. URL http://CRAN.R-project.org/package=shiny. R package version 0.3.0. [p9]

D. Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer-Verlag, New York, 2008. URL http://lmdvr.r-forge.r-project.org. ISBN 978-0-387-75968-5. [p1]

R. Vaidyanathan. *rCharts: Interactive Charts using Polycharts.js*, 2013. R package version 0.3.51. [p8]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, New York, 2009. ISBN 978-0-387-98140-6. URL http://had.co.nz/ggplot2/book. [p1]

Y. Xie. animation: An R package for creating animations and demonstrating statistical methods. *Journal of Statistical Software*, 53:1–27, 2013. URL http://www.jstatsoft.org/v53/i01/. [p8]

*Paul Murrell*
*The University of Auckland*
*Auckland*
*New Zealand* paul@stat.auckland.ac.nz

*Simon Potter*
*The University of Auckland*
*Auckland*
*New Zealand* simon@sjp.co.nz