Gore: What is it Good For?

Paul Murrell

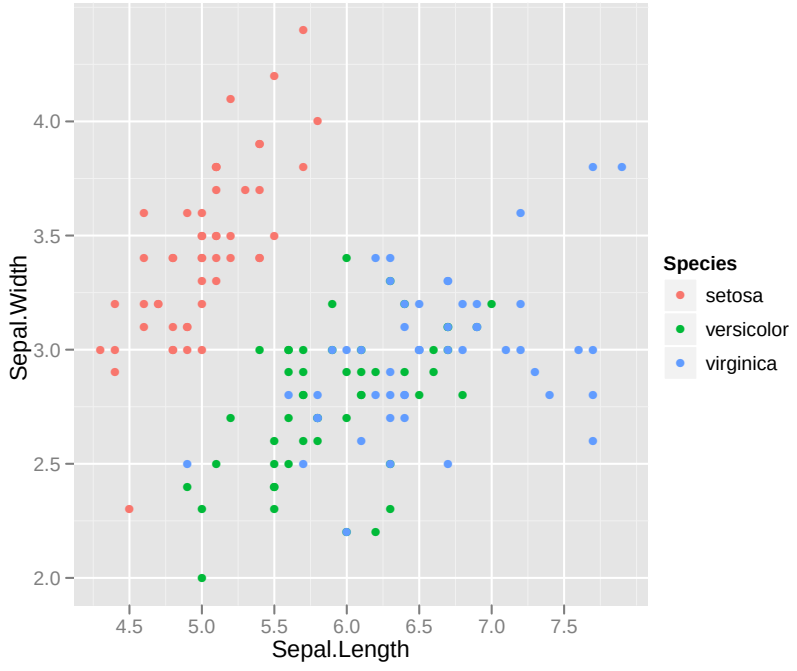
The University of Auckland

July 2011

Overview

- Several R graphics packages provide convenient high-level functions for producing statistical plots.

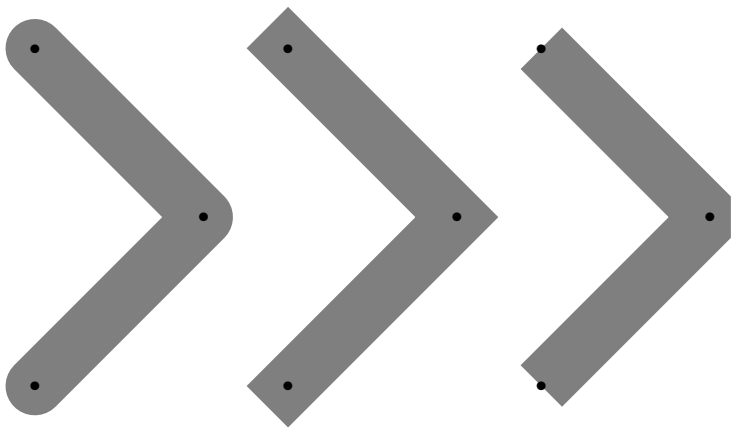
```
library(ggplot2)
qplot(Sepal.Length, Sepal.Width,
      data=iris, color=Species)
```



Overview

- Several graphics packages also provide access to much lower-level graphics facilities (gory graphics details).

```
library(grid)
grid.lines(x, y, gp=gpar(col="grey50", lwd=40,
                         lineend="square",
                         linejoin="mitre"))
```

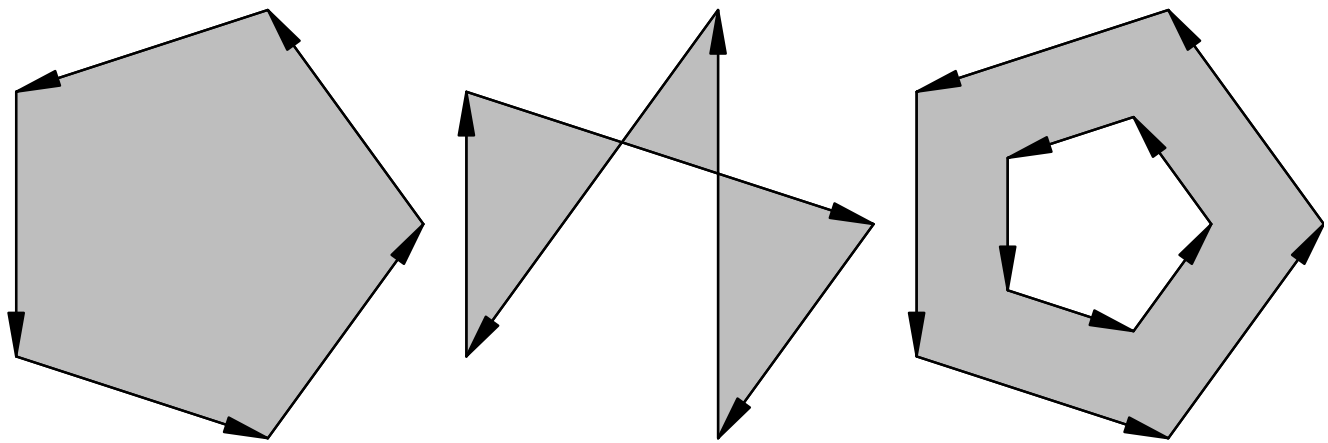


Overview

- What's the point of these gory graphical details?
  - Complex paths and fill-rules
  - Texture mapping
  - Colour spaces
  - Character encodings
  - Line endings

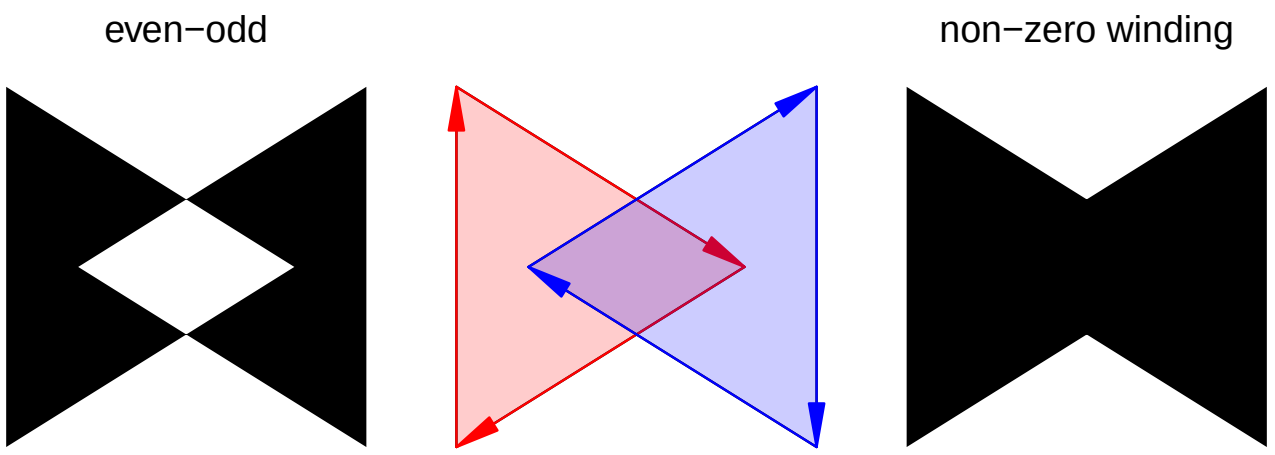
Complex paths

- Sophisticated graphics systems allow for not just general polygonal shapes, but also for complex paths whose edges may self-intersect and/or that may consist of disjoint polygons.



Path fill-rules

- When a polygon describes a complex shape, either with self-intersecting edges, or disjoint paths, there are two standard rules for determining which regions are "inside" the polygon:



Path fill-rules

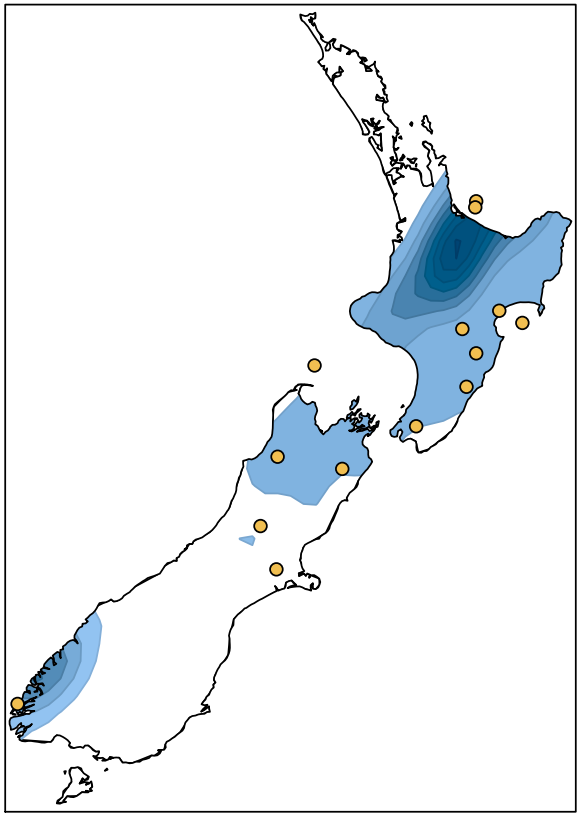
- R provides support for drawing these sorts of complex paths and provides control over the fill-rule.

grid.polygon()

grid.path(rule="winding")

Path fill-rules

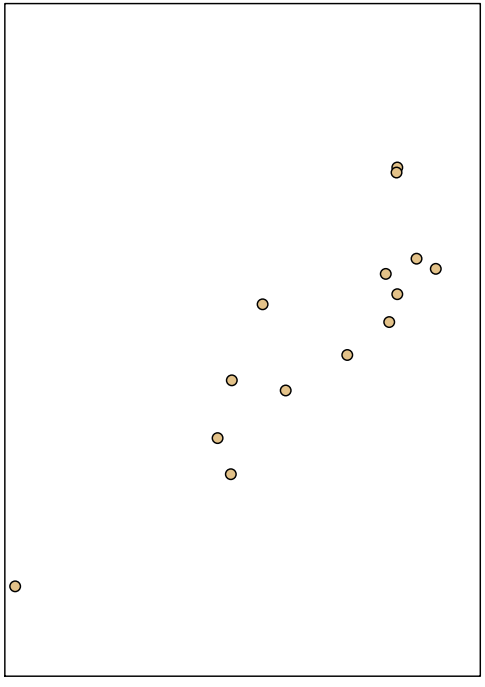
- Complex paths can be useful for drawing plots.



Path fill-rules

- The plot consists of data symbols ...

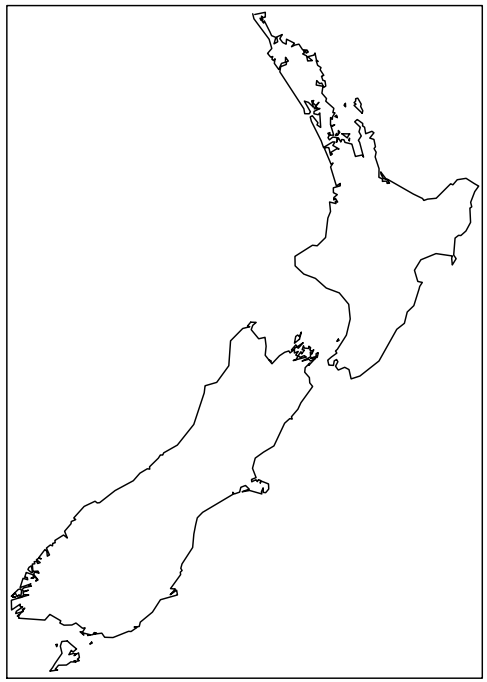
```
points(x, y, ...)
```



Path fill-rules

- ... a map outline ...

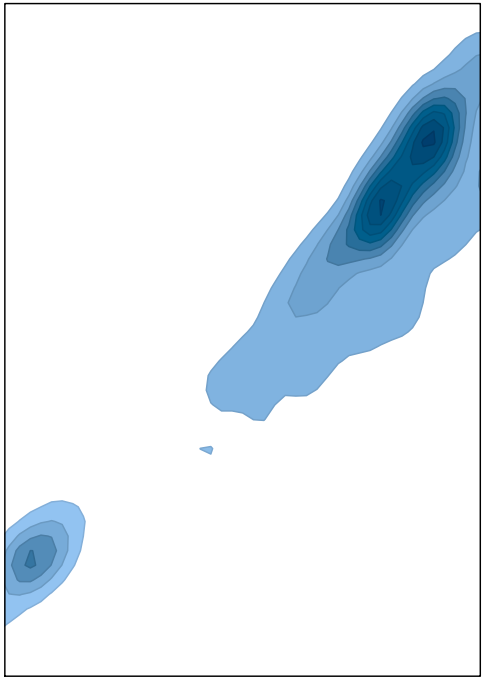
```
library(maps)
map("nz")
```



Path fill-rules

- ... and contour lines.

```
mapply(polygon,
       contourLines(...),
       col=...)
```



Path fill-rules

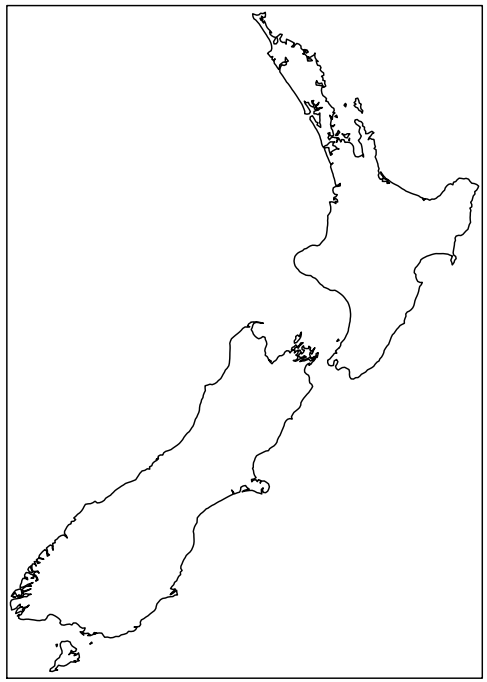
- The map is a complex path that describes the border of New Zealand.

```
outline <- map("nz",
               plot=FALSE)
polypath(outline,
         rule="evenodd")
```

Path fill-rules

- We can add points to add a bounding box to this path ...

```
xrange <- range(outline$x)
xbox <- xrange + c(-2, 2)
path$x <- c(outline$x, NA,
            c(xbox,
              rev(xbox)))
```



Path fill-rules

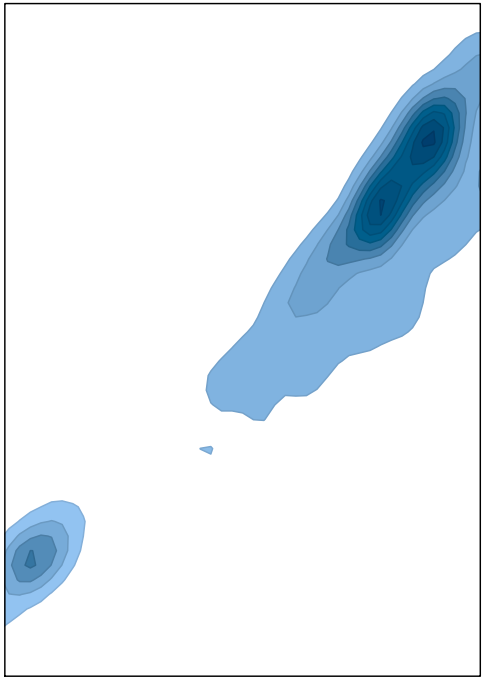
- Filling with an "evenodd" rule now fills everything outside New Zealand.

```
polypath(path,
         rule="evenodd")
```

Path fill-rules

- The contours are drawn first ...

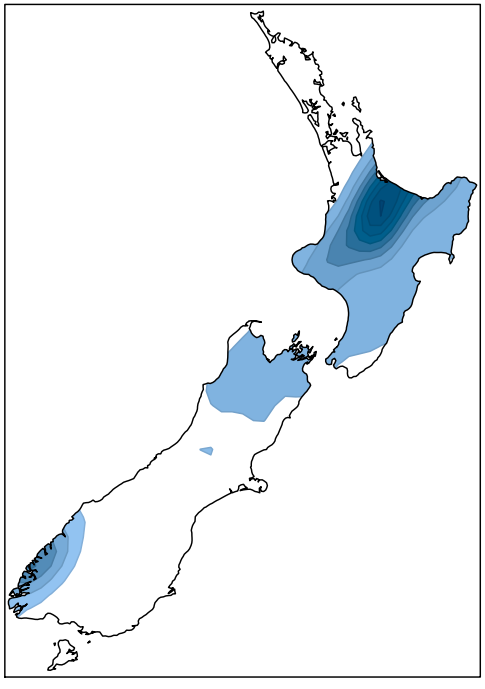
```
mapply(polygon,
       contourLines(...),
       col=...)
```



Path fill-rules

- ... then the inverted New Zealand region is filled to obscure the contours outside New Zealand ...

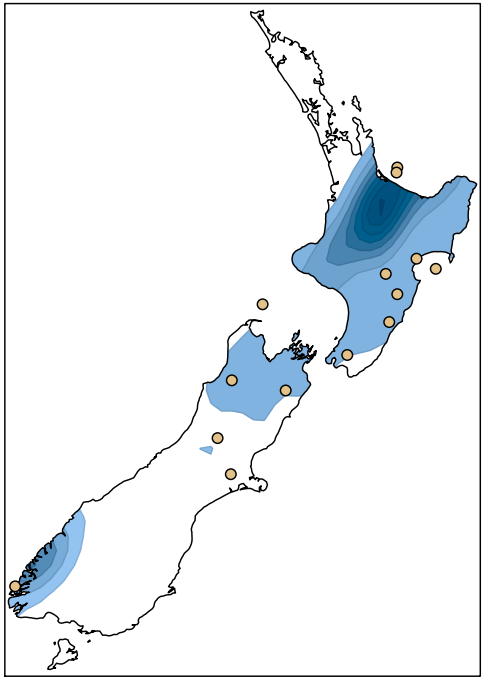
```
mapply(polygon,
       contourLines(...),
       col=...)
polypath(path,
         rule="evenodd")
```



Path fill-rules

- ... and finally the points are drawn over the top.

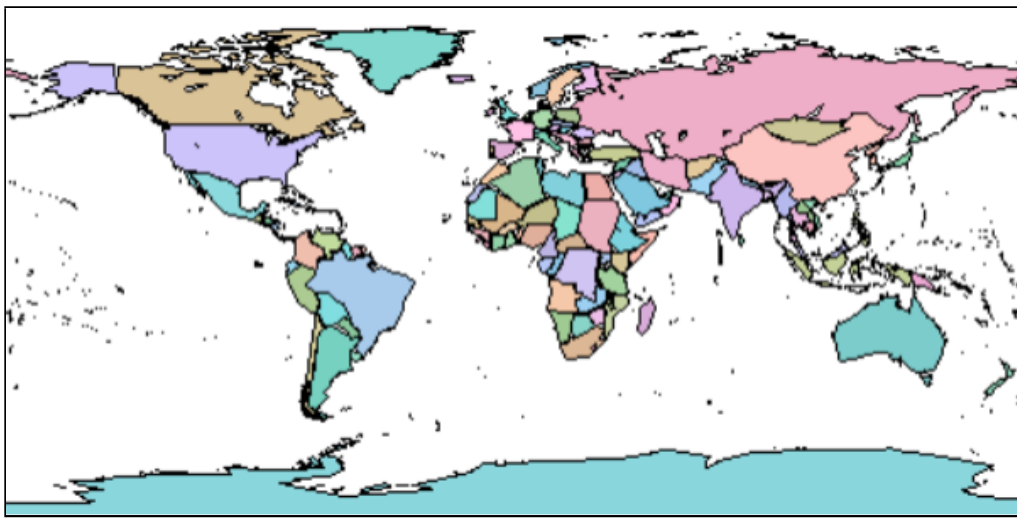
```
mapply(polygon,
       contourLines(...),
       col=...)
polypath(path,
         rule="evenodd")
points(x, y, ...)
```



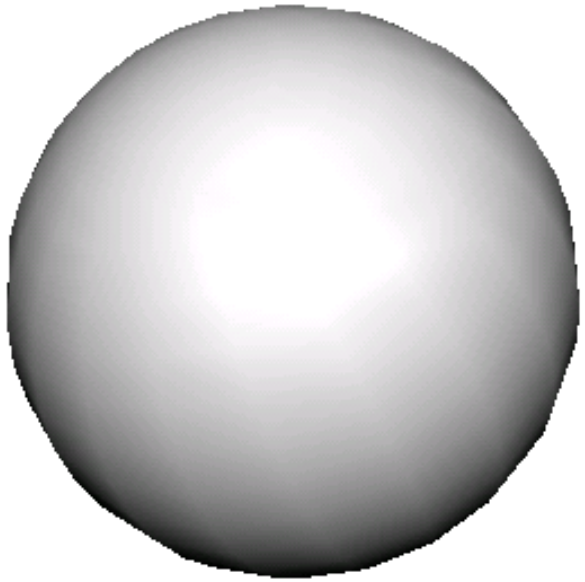
Texture Mapping

- When drawing a 3D image, surfaces can be "coloured" using an image.

```
library(rgl)
```



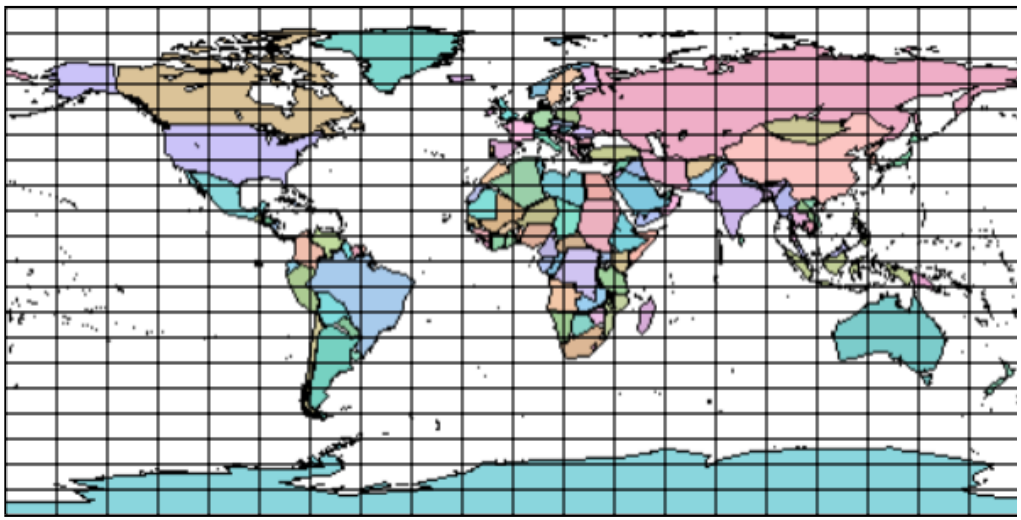
"worldsmall.png"

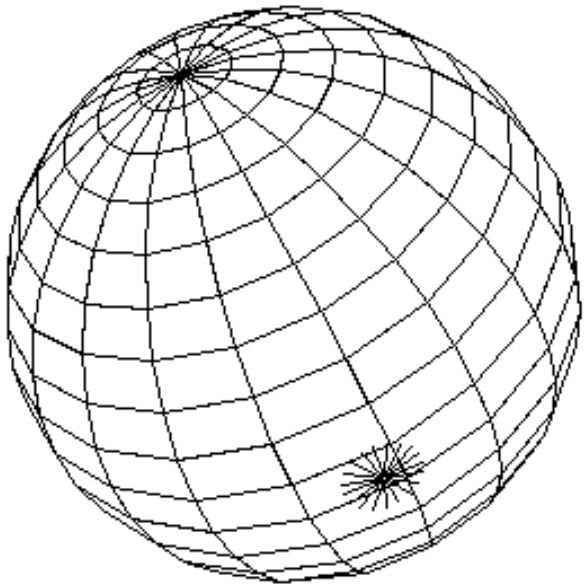


```
persp3d(x, y, z)
```

Texture Mapping

- Points on the image are mapped to vertices on the 3D surface.





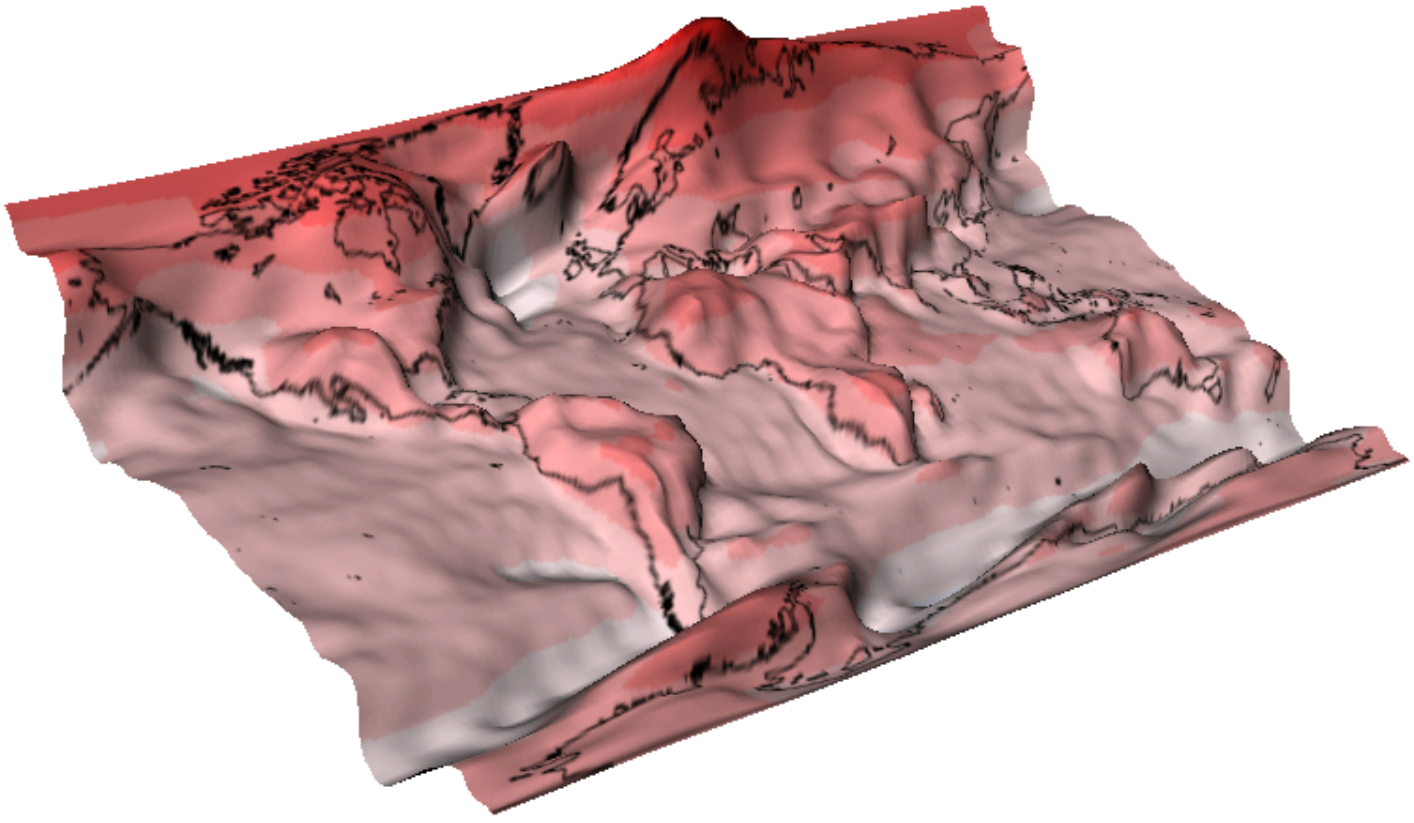
Texture Mapping

- Points on the image are mapped to vertices on the 3D surface.

```
persp3d(x, y, z,
        texture="worldsmall.png")
```

Texture Mapping

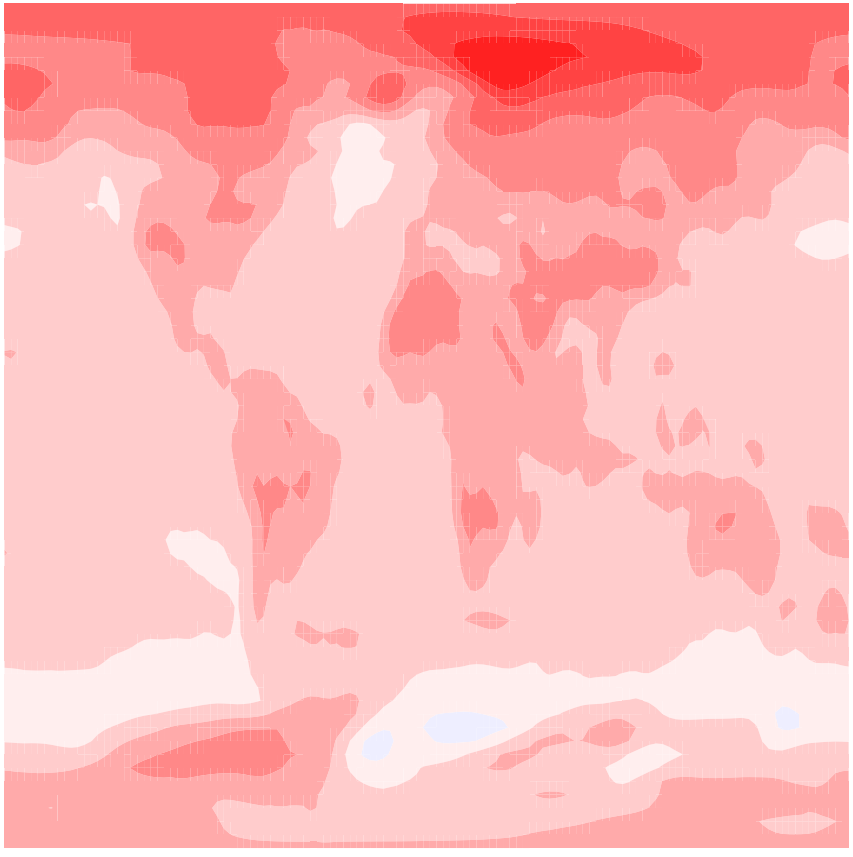
- Texture mapping can be useful for drawing plots.



Texture Mapping

- The plot consists of a filled contour ...

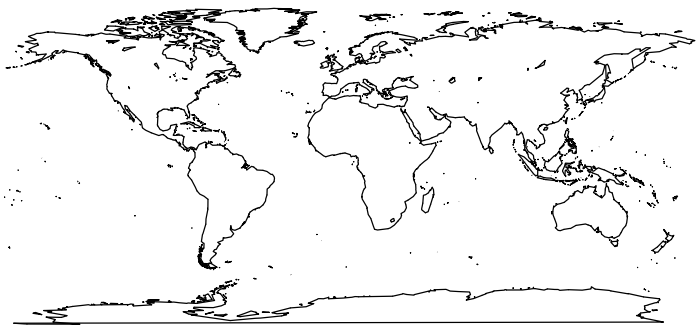
```
filled.contour(x, y, z)
```



Texture Mapping

- ... a map ...

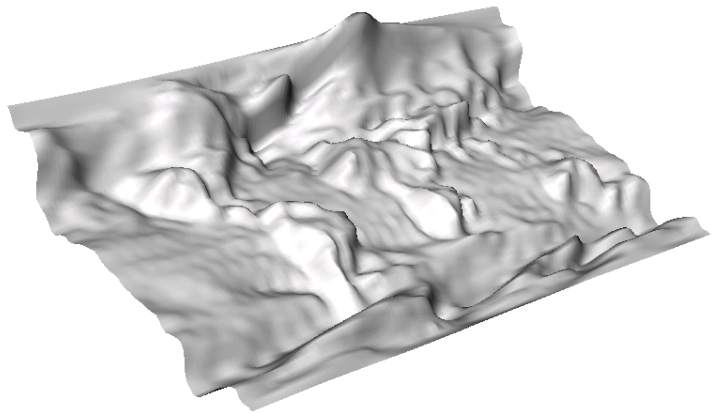
```
library(maps)
map("world")
```



Texture Mapping

- ... and a 3D surface.

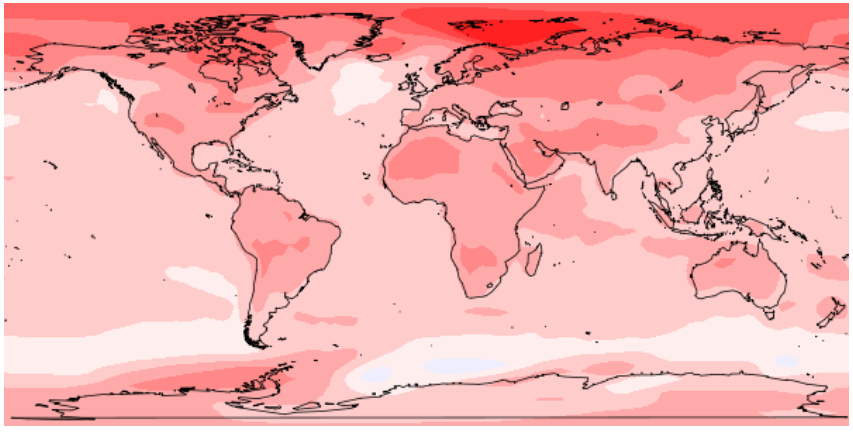
```
library(rgl)
persp3d(x, y, z)
```



Texture Mapping

- The map and heatmap can be combined in a PNG image.

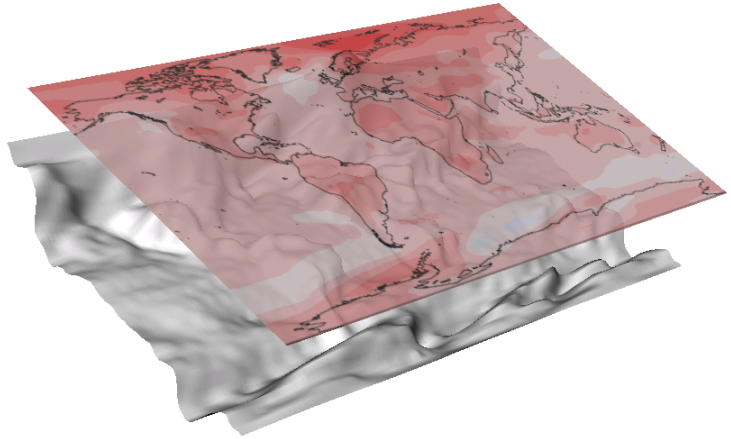
```
png("world.png")
filled.contour(x, y, z)
map("world", add=TRUE)
dev.off()
```

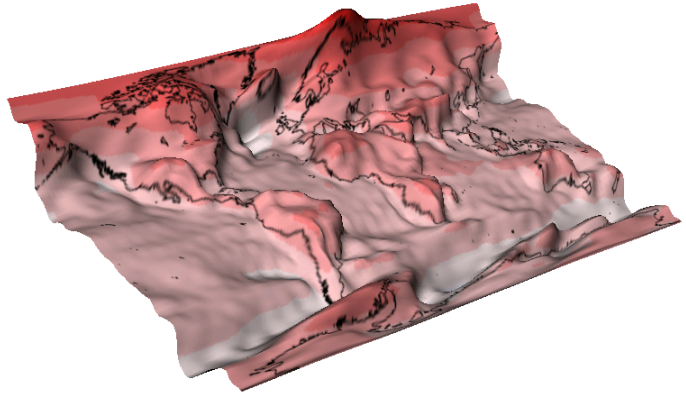


Texture Mapping

- The image is then used as a texture for the surface.

```
persp3d(x, y, x,
        texture="world.png")
```





Colour Spaces

- The colour space that most people are familiar with is RGB (Red Green Blue).

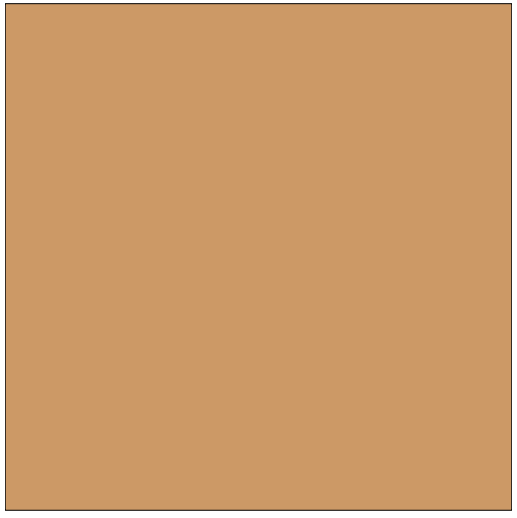
```
rgb(.8, .6, .4)
```

```
[1] "#CC9966"
```

- But this is not a very convenient colour space to work in.
- For example, what colour is produced by the code above?

Colour Spaces

```
grid.rect(gp=gpar(fill=rgb(.8, .6, .4)))
```



Colour Spaces

- A much nicer colour space to work in is HCL (Hue Chroma Lightness).

```
hcl(46, 55, 67)
```

```
[1] "#CC9967"
```

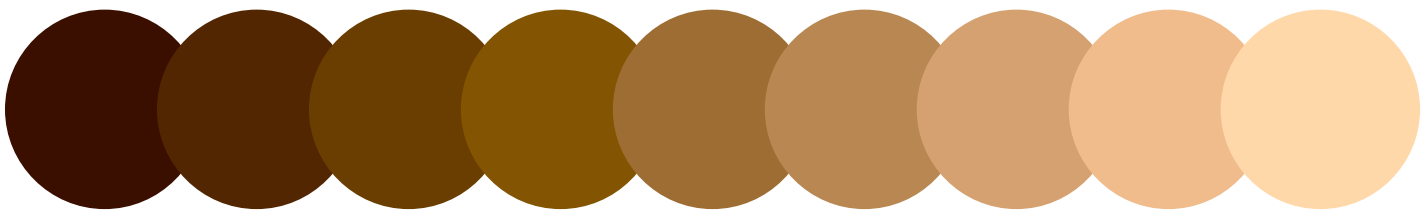
- It is actually still very difficult to intuit the hue (0 yields red, 120 yields green 240 yields blue), but I can at least immediately tell that this is a not very colourful and moderately light colour.

Colour Spaces

- What I also gain is the ability to generate colour **sets** in a rational manner.

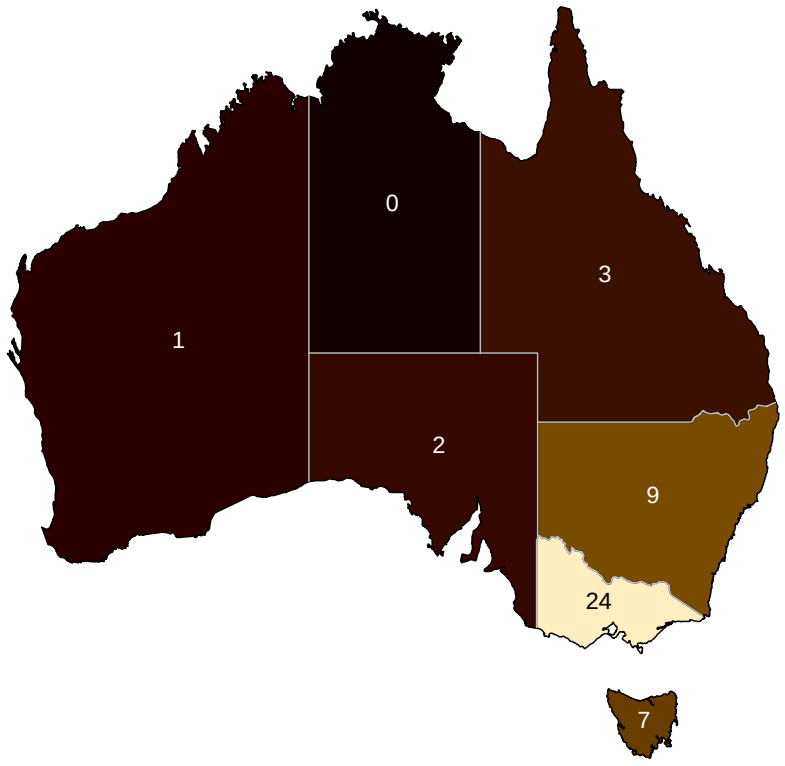
```
hcl(46, 55, seq(10, 90, 10))
```

```
[1] "#3B0F00" "#522600" "#6A3D00" "#835401"
[5] "#9E6D33" "#B98752" "#D5A170" "#F1BC8C"
[9] "#FFD8A9"
```



Color Spaces

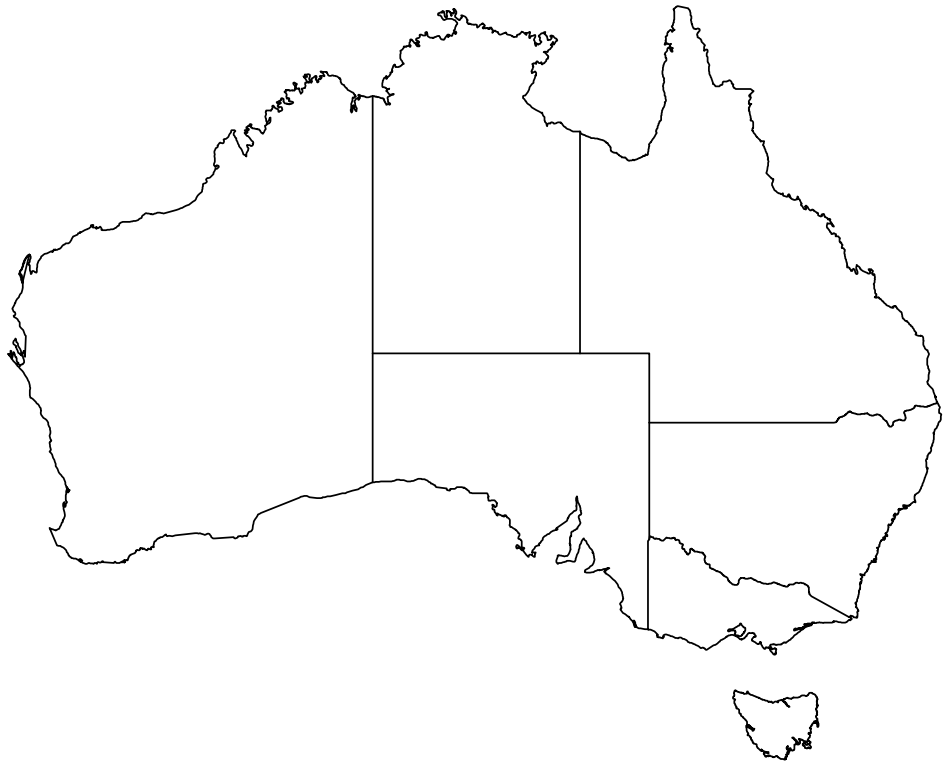
- Color spaces can be useful in plots.



Color Spaces

- The plot consists of polygons describing the states of Australia ...

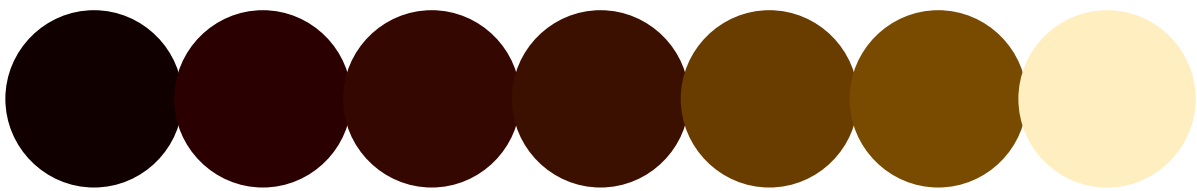
```
library(oz)
oz()
```



Color Spaces

- ... and colours representing the population density of each state.

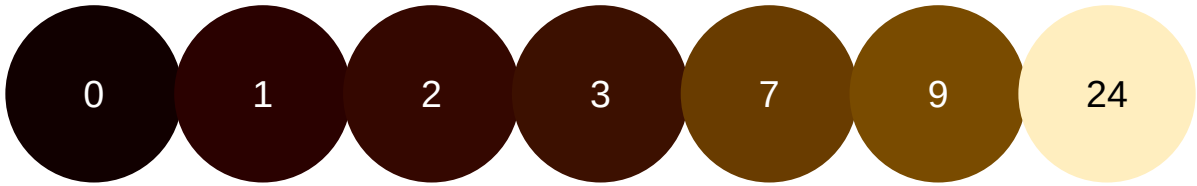
`hcl(46, 55, 4*popDens)`



Color Spaces

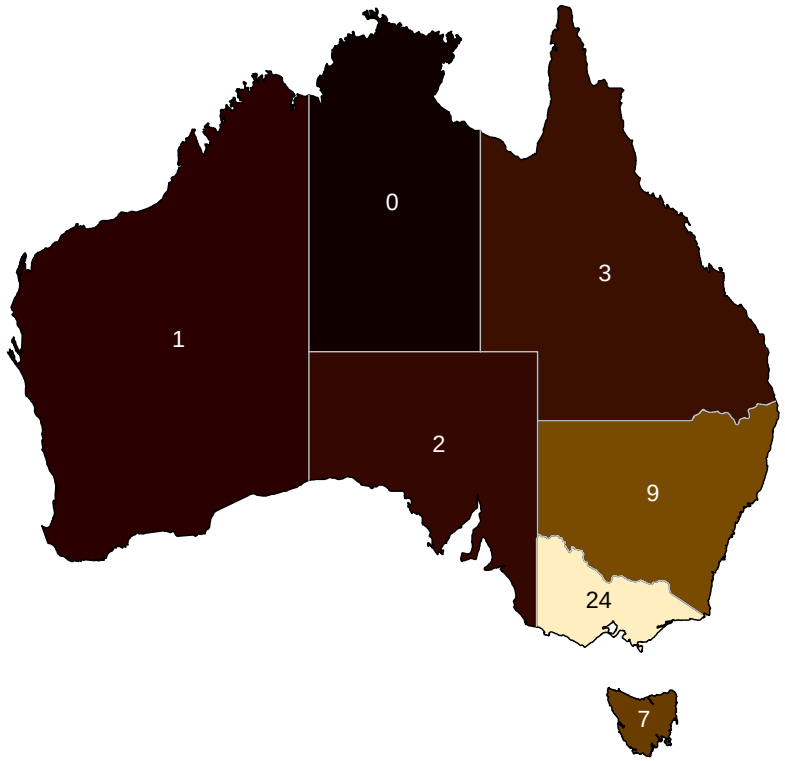
- An additional detail is that the text labels in each state are calculated programmatically from the state colours.

```
ifelse(4*popDens < 80, "white", "black")
```



Color Spaces

```
polygon(australia$states, col=hcl(46, 55, 4*popDens))
text(cntr[1], cntr[2], round(popDens),
     col=ifelse(4*popDens < 80, "white", "black"))
```



Character Encodings

- A **font** is a collection of named glyphs.

/a	→	a
/b	→	b
/c	→	c
/d	→	d
/zero	→	0
/exclam	→	!
/ampersand	→	&
/plusminus	→	±

Character Encodings

- An **encoding** pairs a number code with each named glyph.

141	→	/a	→	a
142	→	/b	→	b
143	→	/c	→	c
144	→	/d	→	d
060	→	/zero	→	0
041	→	/exclam	→	!
046	→	/ampersand	→	&
261	→	/plusminus	→	±

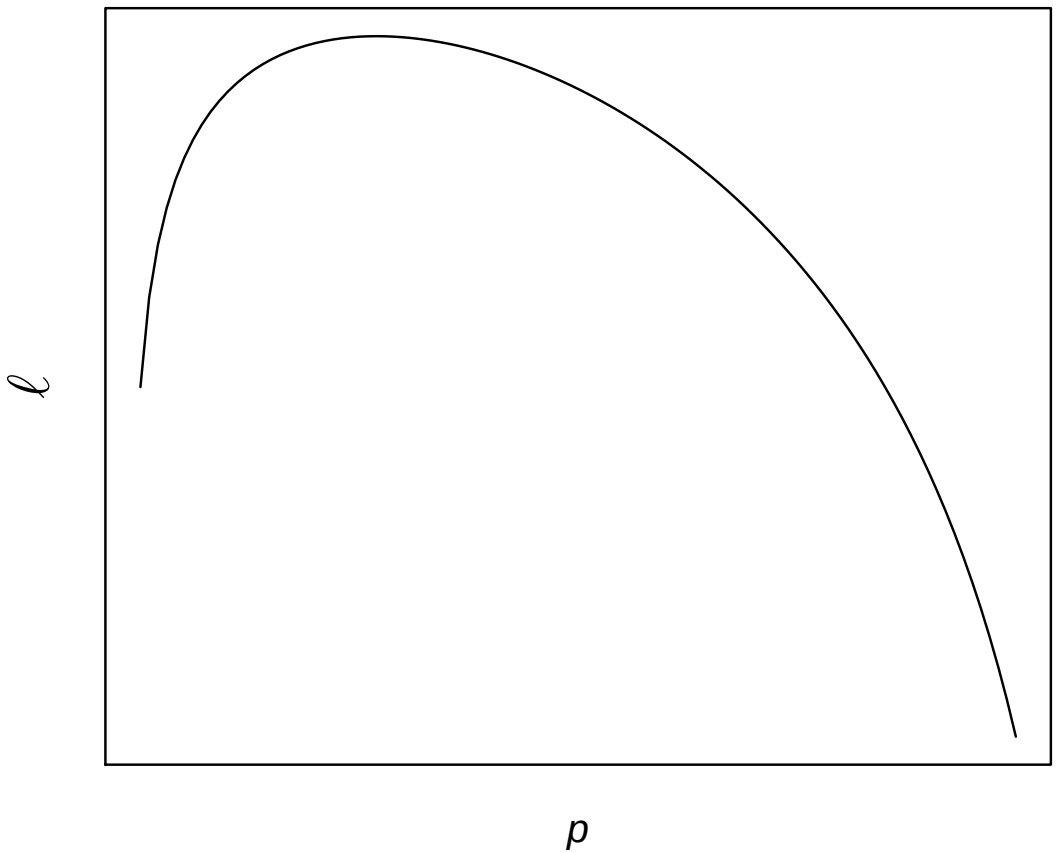
Character Encodings

- Keystrokes are recorded as number codes.
- Special escape sequences can be used to enter number codes that have no corresponding key.

A	→	141	→	/a	→	a
B	→	142	→	/b	→	b
C	→	143	→	/c	→	c
D	→	144	→	/d	→	d
0	→	060	→	/zero	→	0
!	→	041	→	/exclam	→	!
&	→	046	→	/ampersand	→	&
\ 2 6 1	→	261	→	/plusminus	→	±

Character Encodings

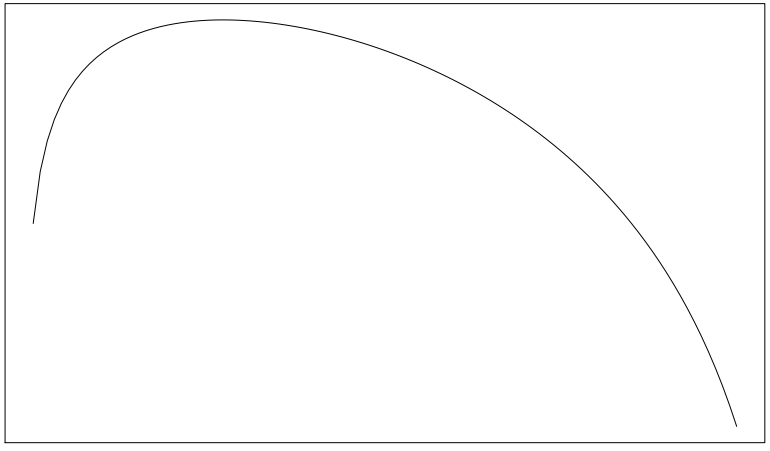
- Character encodings can be useful for drawing plots.



Character Encodings

- The plot consists of a line ...

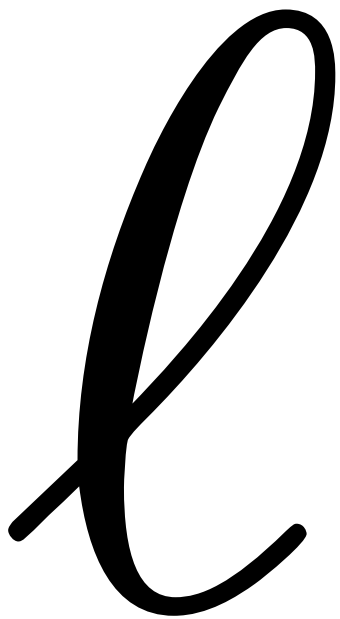
```
curve(20*(3*log(1-x) + log(x)))
```



Character Encodings

- ... and a special script-l character.

```
grid.text("\200", gp=gpar(fontfamily="special"))
```



Character Encodings

- The Computer Modern Math Italic font.

```
...
C 94 ; WX 1000 ; N slurbelow ; B 56 133 943 371 ;
C 95 ; WX 1000 ; N slurabove ; B 56 130 943 381 ;
C 96 ; WX 416.667 ; N lscript ; B 11 -12 398 705 ;
C 97 ; WX 528.588 ; N a ; B 40 -11 498 442 ;
C 98 ; WX 429.165 ; N b ; B 47 -11 415 694 ;
...
```

Character Encodings

- A special encoding file.

```
...
%%  0140
/quoteleft /a /b /c /d /e /f /g
/h /i /j /k /l /m /n /o
/p /q /r /s /t /u /v /w
/x /y /z /braceleft /bar
/braceright /asciitilde /.notdef
%%  0200
/lscript /.notdef /.notdef /.notdef
/.notdef /.notdef /.notdef /.notdef
...
```

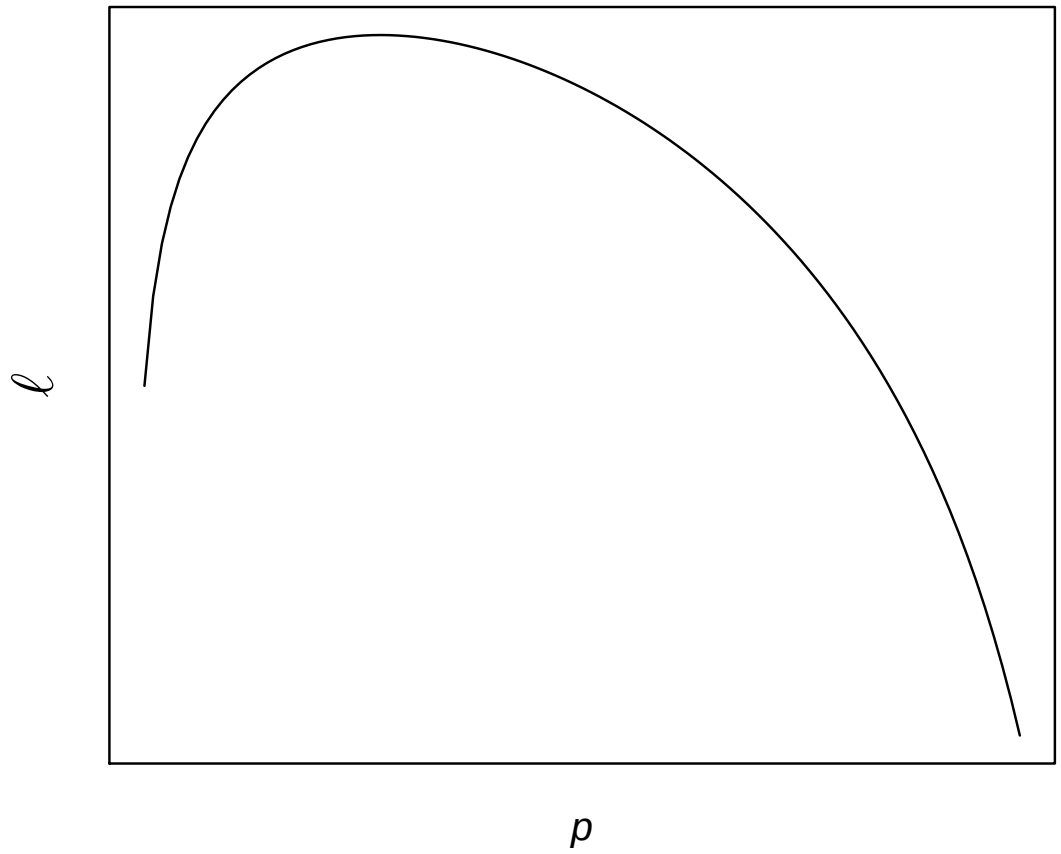
Character Encodings

- Telling R about the font and the encoding.

```
lscriptFont <-
    Type1Font(family="special",
              metrics=c("./cmmi10.afm",
                "Helvetica-Bold.afm",
                "Helvetica-Oblique.afm",
                "Helvetica-BoldOblique.afm"),
              encoding="./special")
pdfFonts(special=lscriptFont)
NA
```

Character Encodings

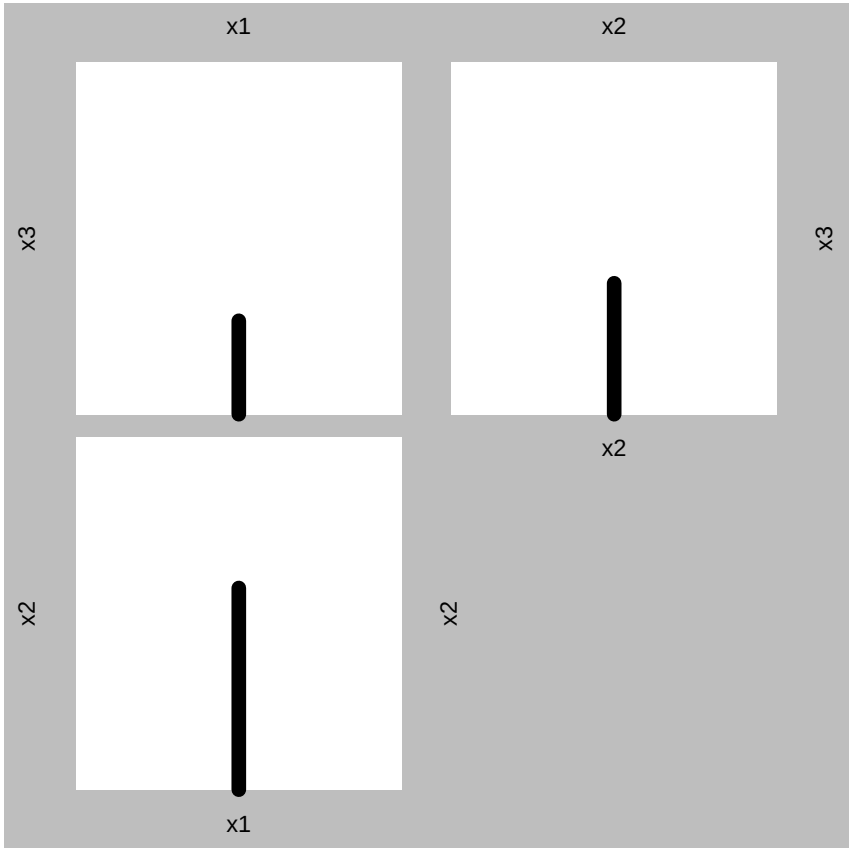
```
curve(20*(3*log(1-x) + log(x)), axes=FALSE)
mtext("\200", side=2, family="special")
```



Line Endings

- The default line ending in R graphics is "round".

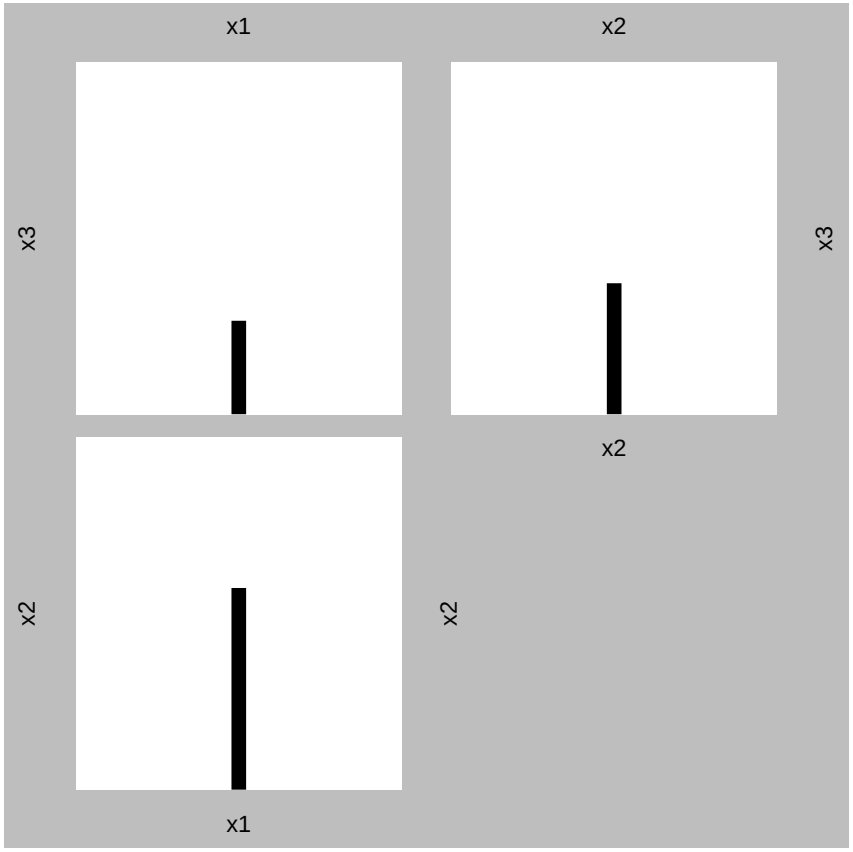
```
segments(.5, 0, .5, runif(1), lwd=10)
```



Line Endings

- Even line endings can be useful for drawing plots.

```
segments(.5, 0, .5, runif(1), lwd=10, lend="butt")
```



Discussion

- As well as producing convenient high-level interfaces for statistical graphics, it is important to retain access to the low-level gory details.
- Implications for software design: the **user** needs access to the gory details.
- Implications for teaching: should we be teaching students the gory details?

Acknowledgements

- The contour map within a map came from discussions with G. Arun Kumar.
- The New Zealand boundaries came from the **maps** package.
- The earthquake data came from the GeoNet Project `http://www.geonet.org.nz/`
- The 3D texture mapping example came from discussion with Claudia Tebaldi.
- The 3D images were produced with the **rgl** package.
- The script-I example came from discussion with Ivo Welch.
- The Australian map boundaries came from the **oz** package.
- Australian state population and area information was obtained from Wikipedia.
- The line endings example was based on an R-help post by Frank Harrell.