Session 02

Lattice Graphics: A closer look

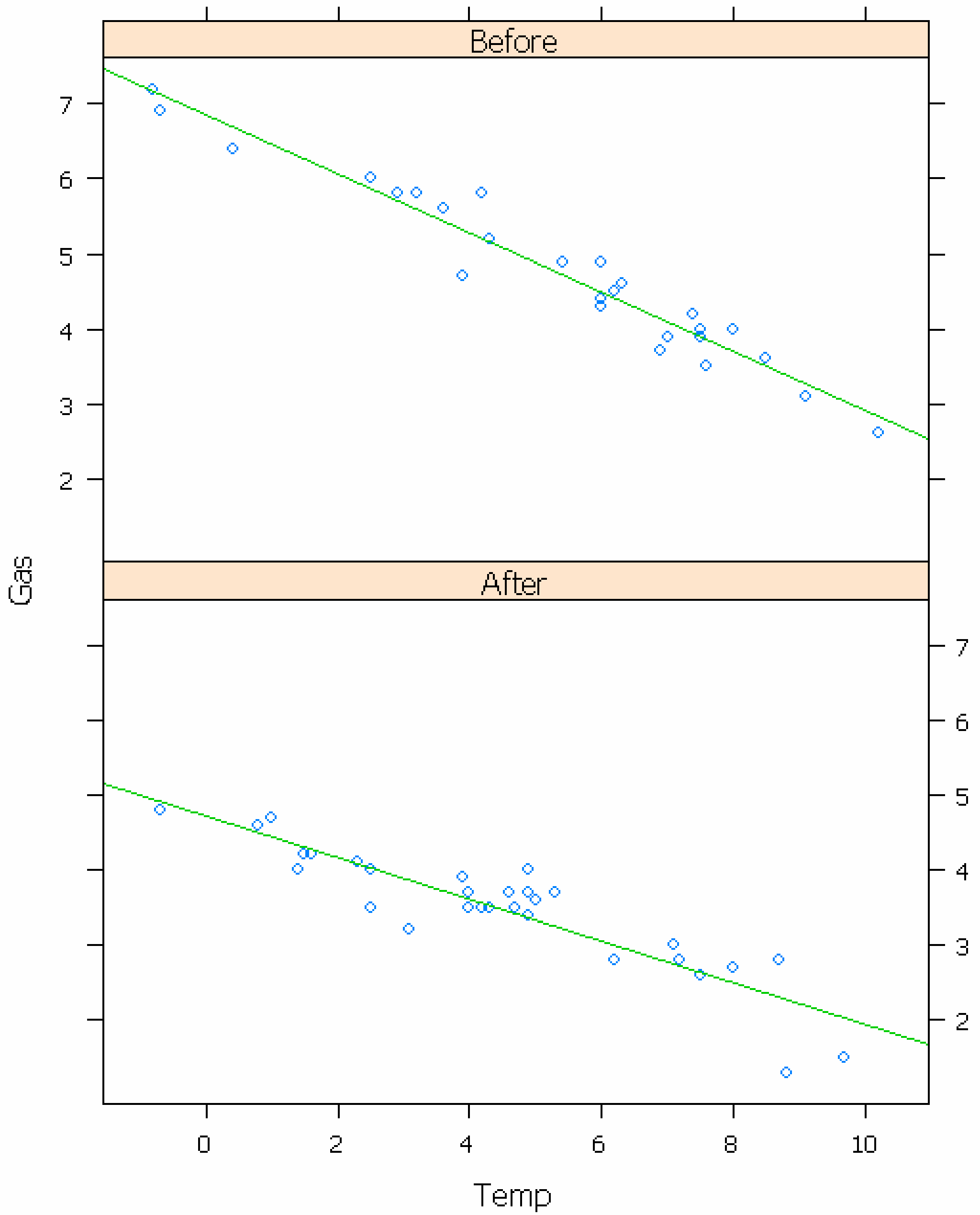
Overview

- Alternative to traditional graphics, but based on it and roughly parallel to it
- Implementing the ideas of Bill Cleveland, and simplest to use when defaults are appropriate
- Incremental construction is not feasible: the entire object must be constructed in a seamless operation
- Functions generate Lattice objects; it is the printing of these objects that produces graphical output.
- Lattice objects may be updated, but this is not often necessary
- List of functions – page 91 of MASS

A simple example: the Whiteside data

```
require(MASS)
names(whiteside)
[1] "Insul" "Temp"  "Gas"
xyplot(Gas ~ Temp | Insul, whiteside,
  panel = function(x, y, ...) {
      panel.xyplot(x, y, ...)
      panel.lmline(x, y, col = 3)
  }, as.table = T, aspect = 0.61)
```

- Readily apparent that insulation reduces the need for heating gas, but the slope is less negative with increasing temperature


Before
After
Gas
Temp
7
6
5
4
3
2
0
2
4
6
8
10

All in one call

```
xyplot(Gas ~ Temp | Insul, whiteside, panel =
  function(x, y, ...) {
      panel.xyplot(x, y, ...)
      panel.lmline(x, y, col=3)
  }, as.table = TRUE, aspect = 0.61,
  ylim = c(0, max(whiteside$Gas)+0.5),
  xlab = "Temperature",
  ylab = "Gas consumption",
  main = "Whiteside data on domestic gas consumption",
  layout = c(2,1))
```

Extras

- Make sure the y-axis has a zero:

```
…, ylim = c(0, max(whiteside$Gas)), …
```

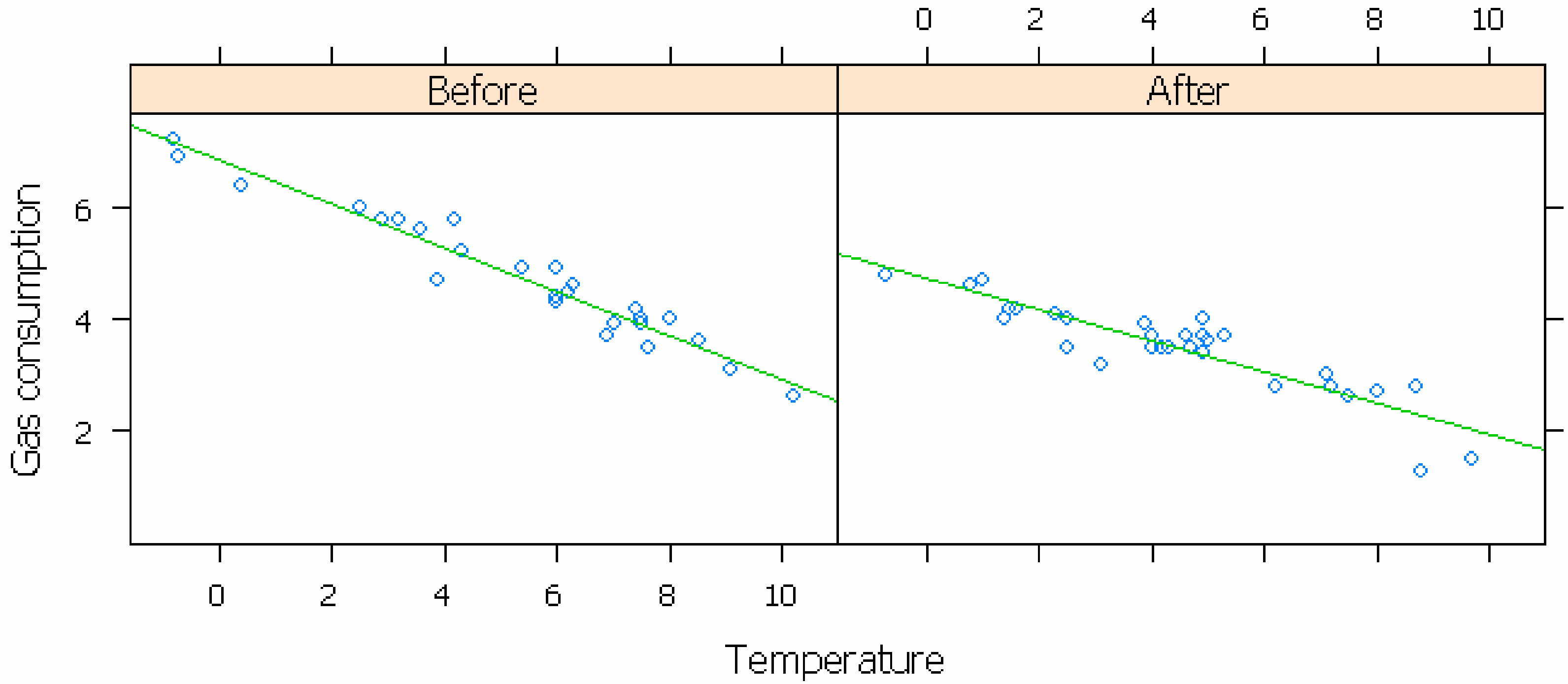
- Give proper labels to the x- and y-axes and add a main label at the top:

```
…, ylab = "Gas consumption",
  xlab = "Temperature",  main =
  "Whiteside data on domestic gas consumption", …
```

- Side-by-side layout:

```
…, layout = c(2,1), …
```

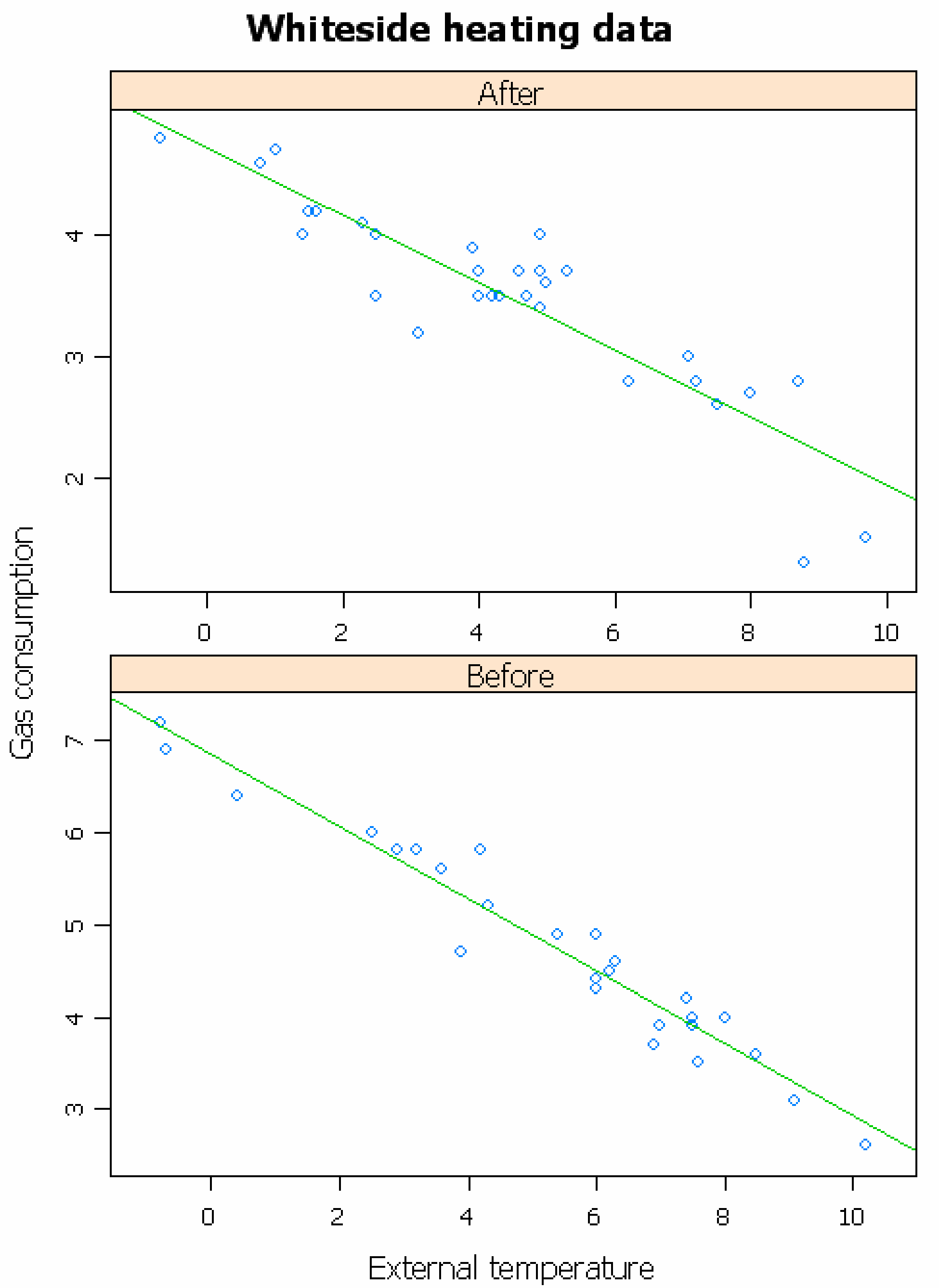
Whiteside data on domestic gas consumption



De-linking the scales

```
xyplot(Gas ~ Temp | Insul, whiteside,
  xlab = "External temperature",
  ylab = "Gas consumption",
  main = "Whiteside heating data", aspect = 0.6,
  panel = function(x, y, ...) {
      panel.xyplot(x, y, ...)
      panel.lmline(x, y, ..., col = 3)
      }, scales = list(relation = "free"))
```

- The plot is now almost misleading! Where possible make full use of linked scales in the panels.


Whiteside heating data
After
Before
Gas consumption
External temperature
0
2
4
6
8
10
2
3
4
5
6
7

Changing Lattice parameters and adding keys

- When a Lattice device is opened, it has various colours, line-types, plotting symbols, &c associated with it that any printing of Lattice objects will use.
- You can change these choices – but it gets messy!
- You need to know how to access this scheme if you want to set a key saying what's what.
- Best to keep an example on hand that works and re-read the help information with this example in mind.

Getting and setting Lattice pars

```
trellis.device(x11)

sps <- trellis.par.get("superpose.symbol")
sps

# _$cex:
# _[1] 0.8 0.8 0.8 0.8 0.8 0.8 0.8
# _
# _$pch:
# _[1] 1 1 1 1 1 1 1
# _
sps$pch <- 1:7

trellis.par.set("superpose.symbol", sps)
```

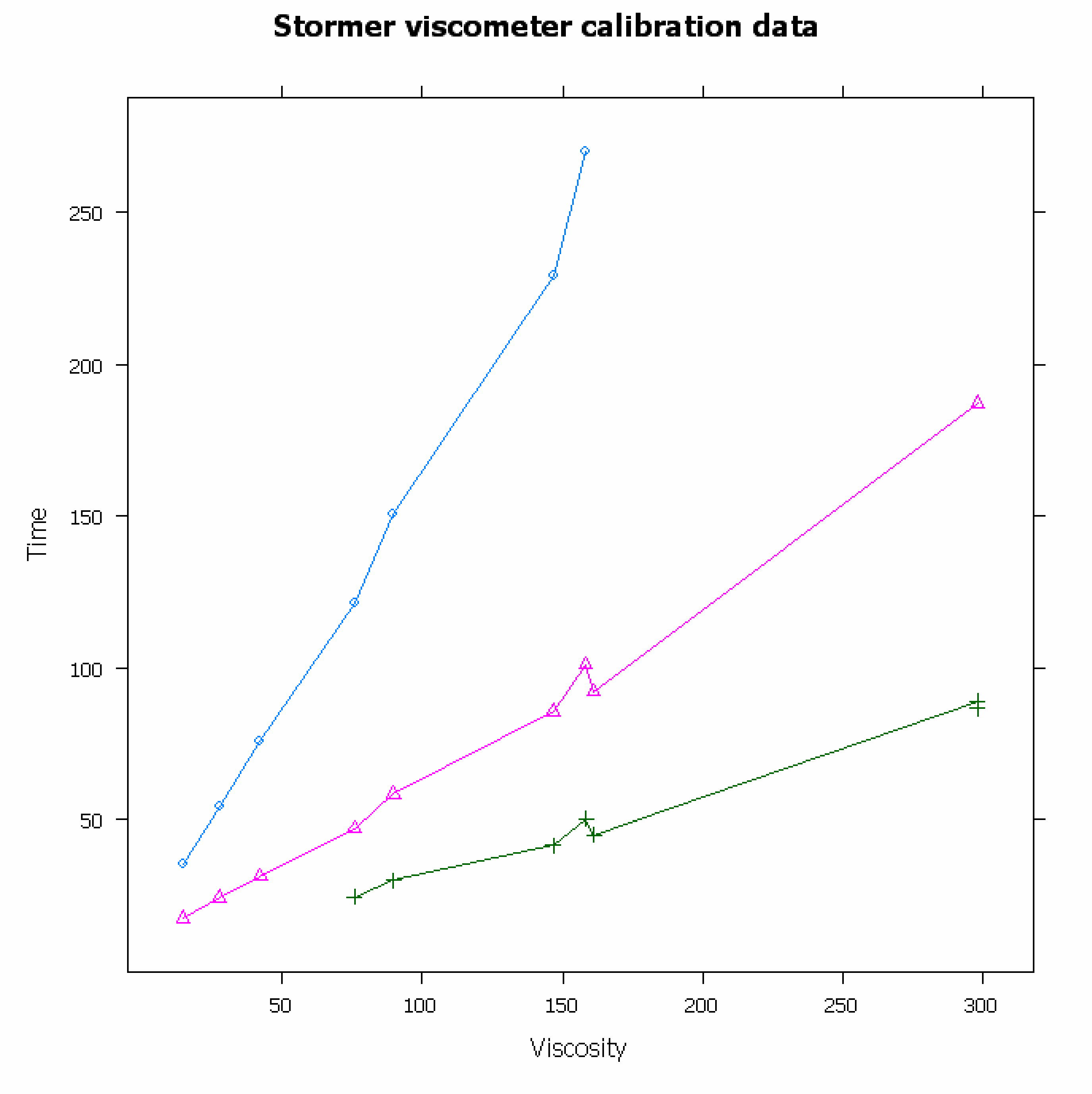
Example: The Stormer viscometer data

- Dependent variable: Time
- Independent variables: Viscosity, Weight
- Theoretical model:

$$T = \frac{\beta V}{W - \theta} + \varepsilon$$

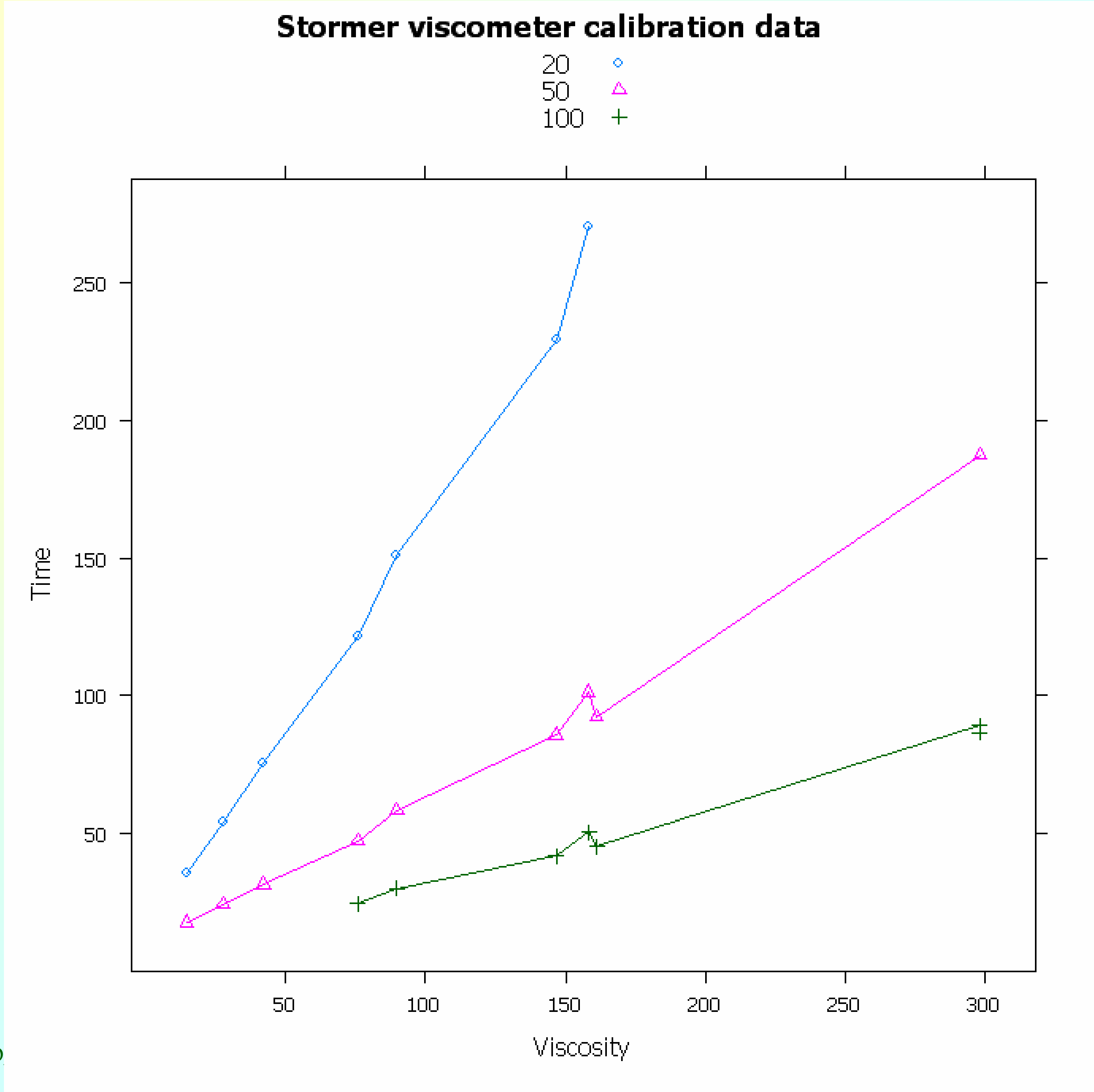
- Plotting Time vs Viscosity should give straight lines with slope depending on Weight.

```
xyplot(Time ~ Viscosity, stormer, groups = Wt,
  type = "b",
  main = "Stormer viscometer calibration data")

dev.off()
```


Stormer viscometer calibration data
Time
250
200
150
100
50
Viscosity
50
100
150
200
250
300

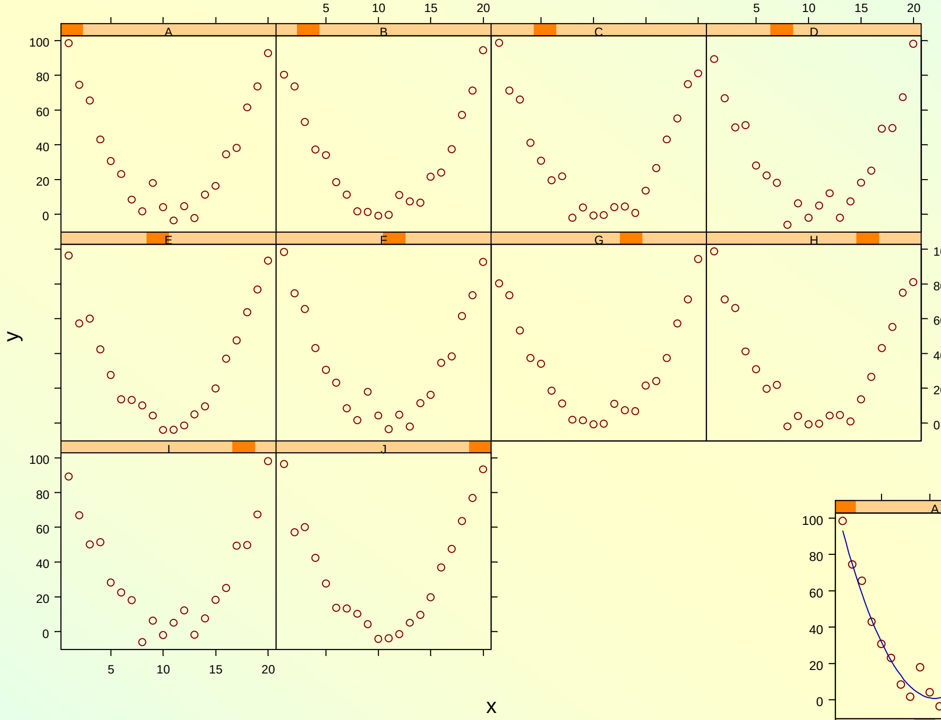
Adding the key

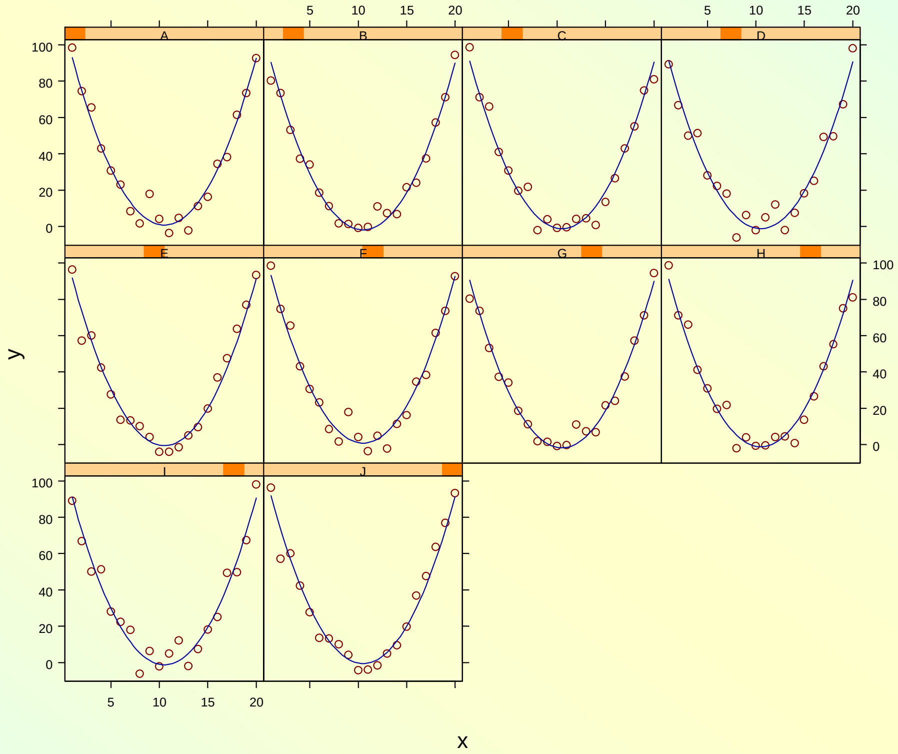
```
xyplot(Time ~ Viscosity, stormer,
  groups = Wt,
  type = "b",
  auto.key = TRUE,
  main =
  "Stormer viscometer calibration data")
```


Stormer viscometer calibration data
20
50
100
Time
250
200
150
100
50
Viscosity
50
100
150
200
250
300

Artificial example: adding fitted values

```
dummy <- data.frame(x = rep(1:20, 10),
                 f = factor(rep(LETTERS[1:10], each = 20)))
dummy$y <- (dummy$x - 10.5)^2 + rnorm(100, sd = 5)
xyplot(y ~ x | f, dummy, as.table = T)

fm <- lm(y ~ f + poly(x, 2), dummy)
dummy$fitted <- fitted(fm)

xyplot(y ~ x | f, dummy, as.table = T, subscripts = T,
   prepanel = function(x, y, subscripts, ...) {
        list(xlim = range(x),
        ylim = range(y, dummy$fitted[subscripts]),
        dx = diff(range(x)), dy = diff(range(y)))
        },
   panel = function(x, y, subscripts, ...) {
        panel.xyplot(x, y, ...)
        lines(spline(x, dummy$fitted[subscripts]), col = 3)
   }, aspect = "xy")
```

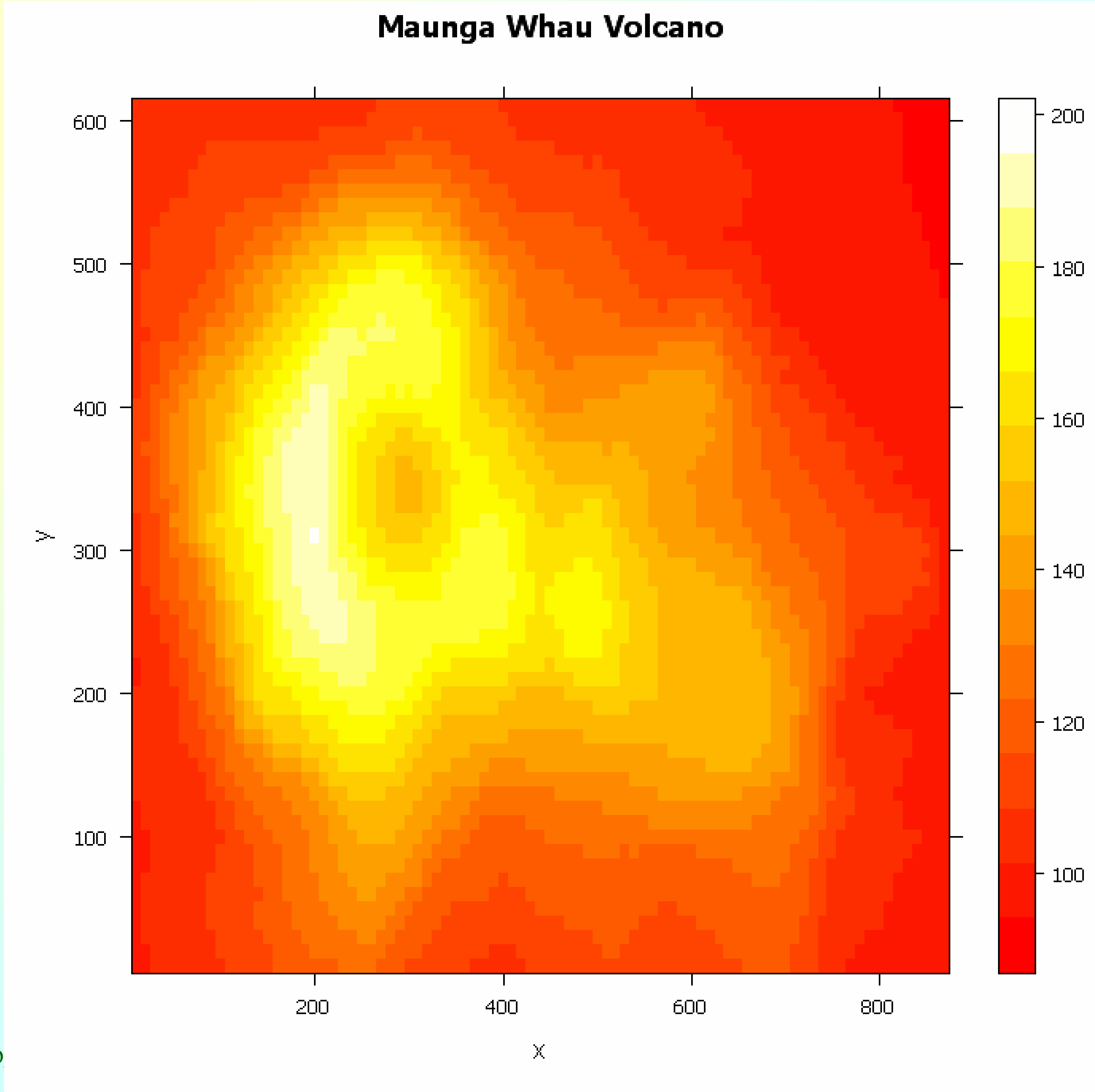

A
B
C
D
E
F
G
H
I
J
y
x
0
20
40
60
80
100
5
10
15
20

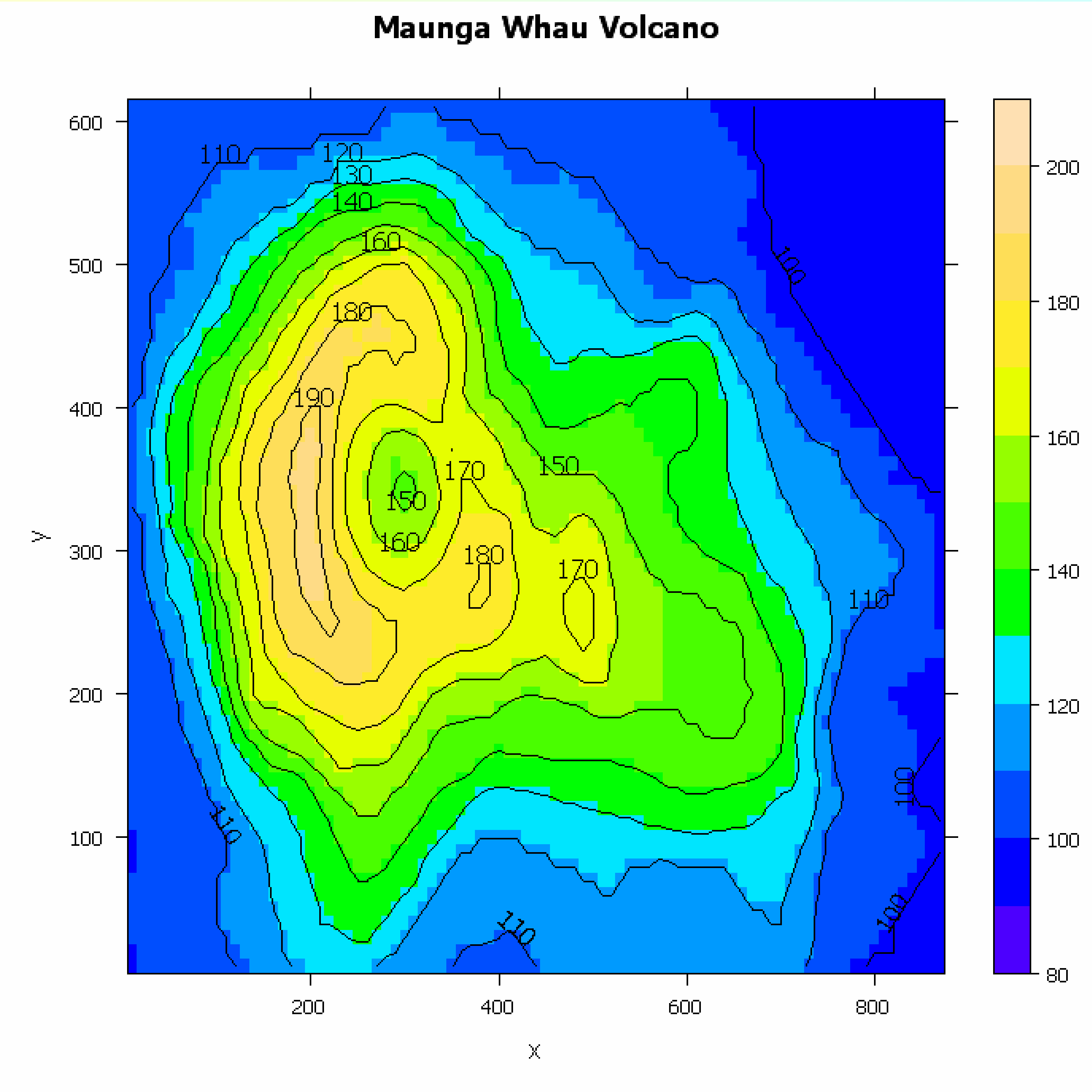

A
B
C
D
E
F
G
H
I
J
y
x
0
20
40
60
80
100
5
10
15
20

A three-dimensional example

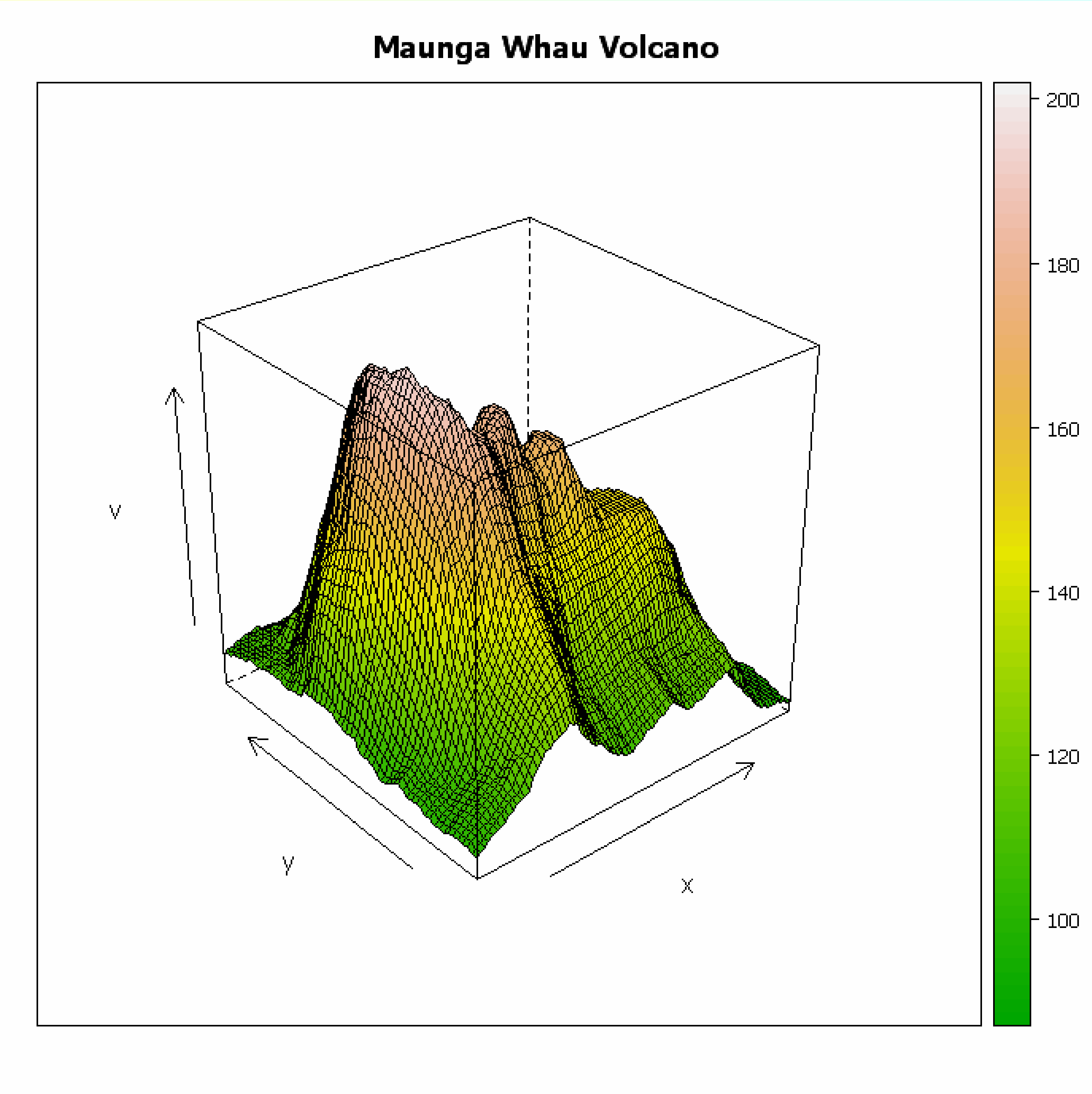
```
find("volcano.S")  # courtesy Ross Ihaka
[1] "package:datasets"
class(volcano)
[1] "matrix"
> dim(volcano)
[1] 87 61

# volcano <- as.matrix(volcano)
x <- 10*(1:nrow(volcano))
y <- 10*(1:ncol(volcano))
vdat <- transform(expand.grid(x = x, y = y),
                  v = as.vector(volcano))

image.palette(heat.colors(256))

levelplot(v ~ x*y, vdat, main = "Maunga Whau Volcano")

image.palette(topo.colors(256))

wireframe(v ~ x*y, vdat, drape = T, main = "Maunga Whau Volcano")
```

```
levelplot(v ~ x*y, vdat,
  col.regions = heat.colors(256),
  main = "Maunga Whau Volcano")

levelplot(v ~ x*y, vdat, contour = TRUE,
  pretty = TRUE,
  col.regions = topo.colors(256),
  main = "Maunga Whau Volcano")

wireframe(v ~ x*y, vdat, drape = TRUE,
  col.regions = terrain.colors(256),
  main = "Maunga Whau Volcano")
```


Maunga Whau Volcano
600
500
400
300
200
100
y
200
400
600
800
x
200
180
160
140
120
100



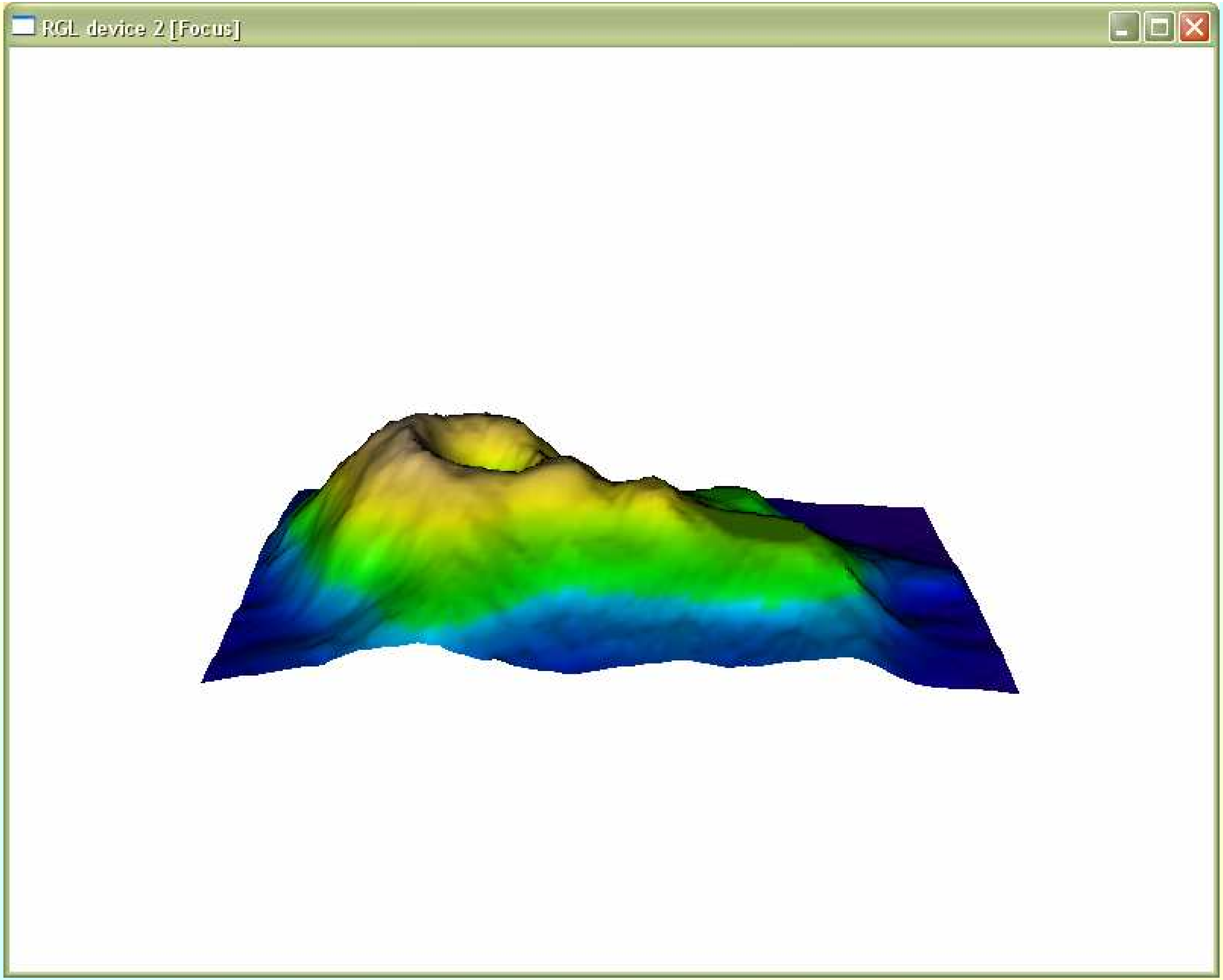
Maunga Whau Volcano
600
500
400
300
200
100
y
200
400
600
800
x
110
120
130
140
160
180
190
170
150
160
180
170
150
100
110
110
100
110
100
200
180
160
140
120
100
80

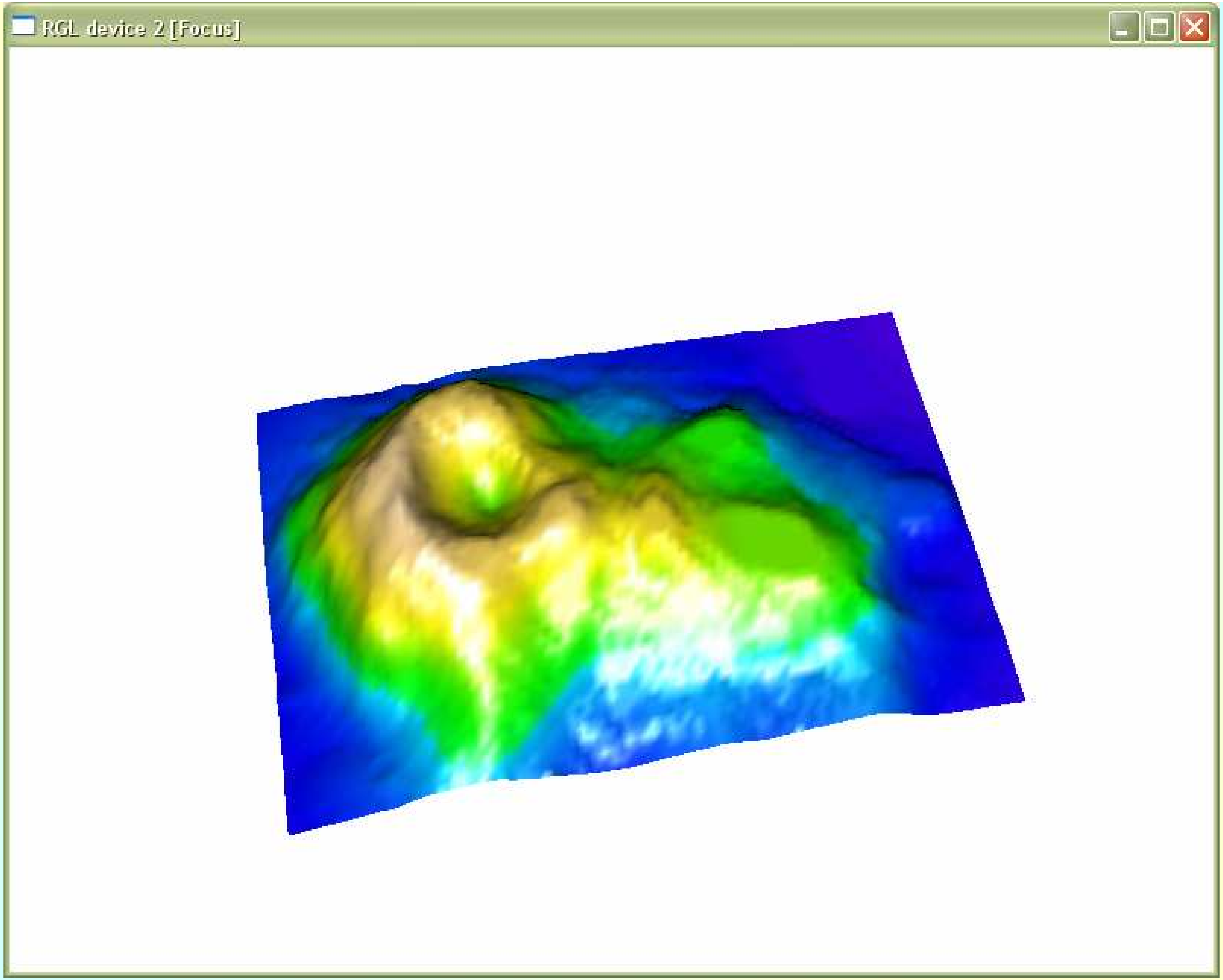


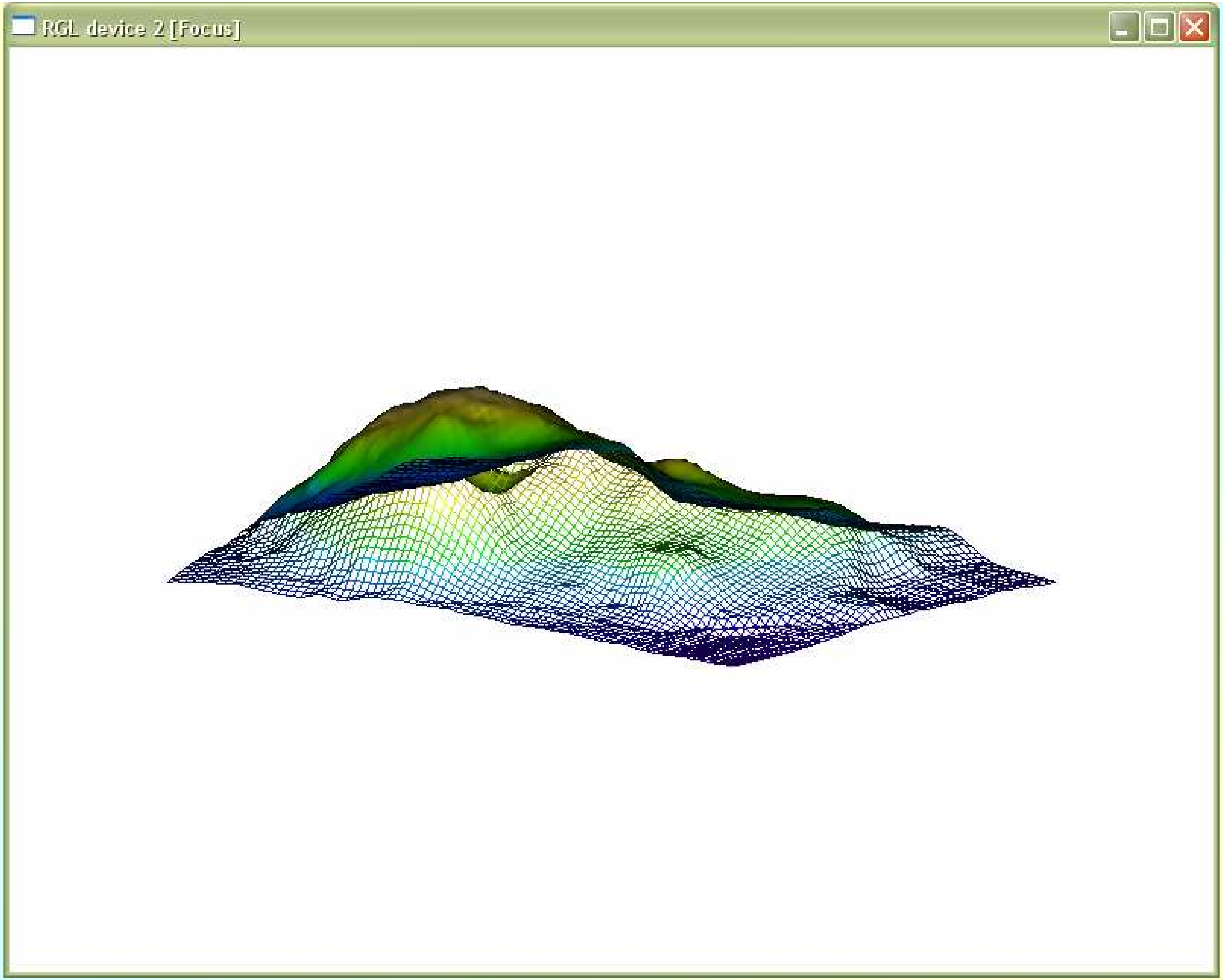
Maunga Whau Volcano
v
y
x
200
180
160
140
120
100

Dynamic versions

```
require(rgl)
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)

zlim <- range(z)
zlen <- zlim[2] - zlim[1] + 1

colorlut <- topo.colors(zlen) # color lookup table

col <- colorlut[ z-zlim[1]+1 ]
# assign colors to heights for each point

open3d()
surface3d(x, y, z, color=col, back="lines")
```


RGL device 2 [Focus]


RGL device 2 [Focus]


RGL device 2 [Focus]